MASTER EN SCIENCES INFORMATIQUES À FINALITÉ SPÉCIALISÉE EN SOFTWARE ENGINEERING

Application du proactive computing

Adaptation automatique des règles d'accès aux données et services dans une architecture IoT

Picard, Noé

Award date:
2018

Awarding institution:
Universite de Namur

Université de Namur
Faculté d'informatique
Année académique 2017–2018

Application du proactive computing : Adaptation automatique des règles d'accès aux données et services dans une architecture IoT

Noé Picard

Maître de stage : Denis Zampunieris

Promoteur : _____ (Signature pour approbation du dépôt - REE art. 40)
Jean-Noël Colin

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

Résumé

Le modèle de contrôle d'accès ABAC a été créé avec comme principale caractéristique de fournir une flexibilité accrue, en se basant sur des politiques combinant des attributs provenant de diverses sources. Le paradigme de l'Internet des Objets se répandant de plus en plus à travers le monde, la possibilité d'utilisation de systèmes équipés de capteurs et d'actionneurs s'étend également. Dans ce contexte, une combinaison entre le modèle ABAC et l'Internet des Objets pourrait théoriquement être bénéfique pour deux raisons, (1) car cela permettrait d'étendre la richesse des attributs disponibles à la rédaction de politiques de sécurité, (2) tout en offrant la possibilité d'agir sur l'environnement lorsque c'est nécessaire. Ce mémoire propose l'utilisation du proactive computing, à travers un moteur proactif, afin de rendre possible cette combinaison.

Mots-clés : Internet des Objets, Contrôle d'accès, ABAC, Proactive Computing

Abstract

The ABAC access control model was created with the main feature of providing increased flexibility, by using policies that can combine attributes from various sources. The paradigm of the Internet of Things is spreading more and more throughout the world, the possibility of using systems equipped with sensors and actuators has therefore also increased. In this context, a combination between the ABAC model and the Internet of Things could theoretically be beneficial for two reasons, (1) because it would extend the wealth of available attributes to the drafting of security policies, (2) while offering the possibility to act on the environment when necessary. This thesis proposes the use of proactive computing, through a proactive engine, to make this combination possible.

Keywords : Internet of Things, Access Control, ABAC, Proactive Computing

Préface

Ce mémoire est présenté en vue de l'obtention du grade de Master en Sciences informatiques à l'Université de Namur. Il constitue l'aboutissement d'un travail dont les prémices remontent à septembre 2017, lors de mon stage à l'Université du Luxembourg qui s'est déroulé sous la supervision de mon maître de stage, le Professeur Denis Zampunieris.

Mes premiers remerciements vont au Professeur Zampunieris pour son accueil chaleureux, ainsi que pour son suivi et ses conseils avisés tout au long du stage. Il a toujours su se rendre disponible tout en laissant une réelle autonomie dans les travaux effectués, me permettant ainsi de faire l'expérience authentique du milieu de la recherche académique.

Je tiens également à remercier mon promoteur, le Professeur Jean-Noël Colin, pour son implication et ses précieux conseils durant le stage, et qui m'a épaulé durant la rédaction de ce mémoire, par sa disponibilité et ses remarques judicieuses.

Enfin, je remercie ma famille pour leur patience et leur relecture attentive avant la remise de ce mémoire, ainsi que mes amis et camarades de classe pour le soutien mutuel que nous nous sommes partagés.

Noé Picard
Mai 2018

Table des matières

Table des figures

Liste des tableaux

Liste des listings

Acronymes

6LoWPAN IPv6 Low power Wireless Personal Area Networks.

ABAC Attribute-Based Access Control.

ACL Access Control List.

DAC Discretionary Access Control.

IoT Internet of Things.

IPv6 Internet Protocol version 6.

MAC Mandatory Access Control.

MLS Multilevel Security.

MQTT Message Queue Telemetry Transport.

PAP Policy Administration Point.

PDP Policy Decision Point.

PEP Policy Enforcement Point.

PIP Policy Information Point.

PRP Policy Repository Point.

RBAC Role-Based Access Control.

TCP Transmission Control Protocol.

XACML eXtensible Access Control Markup.

Introduction

Le contrôle d'accès joue une part importante dans la sécurité des systèmes d'information existants, aussi bien pour les accès physiques que logiciels. Dans sa définition générale, il vise à réguler les accès qu'un sujet peut effectuer sur une ressource. Cela peut donc par exemple concerner la lecture d'un fichier ou l'accès à une pièce d'un bâtiment. Il existe différents modèles pour appliquer une politique de sécurité, tels que le modèle de contrôle d'accès discrétionnaire DAC ou non discrétionnaire MAC, ainsi que le modèle RBAC à base de rôles. Mais plus récemment, le modèle ABAC émerge de plus en plus, notamment grâce à sa flexibilité et son aspect dynamique. Ce modèle utilise le concept d'attribut pour construire les politiques permettant d'appliquer le contrôle d'accès. La nature d'un attribut est très variable, puisqu'il peut s'agir de caractéristiques à propos du sujet, de la ressource ou encore de l'environnement.

Pour permettre un contrôle d'accès encore plus efficace, il semble intéressant de combiner le modèle ABAC avec un paradigme qui a beaucoup gagné en popularité ces dernières années : l'Internet des Objets, souvent référencé par l'acronyme IoT. Celui-ci est considéré par certains comme une nouvelle révolution d'Internet, parfois qualifié de Web 3.0. L'idée principale est d'étendre la connectivité à des choses du monde physique pouvant se trouver partout autour de nous. Cela permet entre autres de récolter des informations sur l'environnement, mais aussi d'agir sur celui-ci. Ainsi, dans le cadre d'une combinaison entre le modèle ABAC et un système IoT, l'intérêt serait d'utiliser ces informations, ou des informations dérivées, dans les politiques de contrôle d'accès. Nous exploiterions donc au maximum les forces du modèle ABAC, en améliorant la richesse des attributs disponibles à l'écriture des politiques de sécurité. De plus, puisque l'IoT nous le permet, agir sur l'environnement conjointement au contrôle d'accès fournirait un avantage non négligeable.

Cette combinaison est l'aspect central de ce mémoire, qui vise à détailler une solution permettant de la concrétiser. Pour y arriver, il a été proposé l'utilisation d'un troisième concept fondamental à la démarche : le proactive computing. Il se définit comme la transition d'une informatique centrée sur l'humain, vers une informatique supervisée par l'humain et parfois non supervisée. Théoriquement défini par David Tennenhouse au début des années 2000, ce concept a trouvé une réalisation à travers le moteur proactif développé par le Professeur Denis Zampunieris et son équipe à l'Université du Luxembourg.

Ce moteur proactif peut nous permettre de contrer de manière efficace les deux obstacles qui s'interposent selon nous à la combinaison dont il est ici question.

Premièrement, il s'agit de trouver un moyen de combler le lien manquant entre le modèle ABAC et un système IoT, qui permettrait d'utiliser les caractéristiques de l'environnement, pouvant être récupérées grâce aux capteurs d'un système IoT, en tant qu'attributs d'un point de vue du modèle ABAC. Deuxièmement, l'interaction avec les actionneurs d'un système IoT étant souhaitée, il est également nécessaire de concilier ce besoin avec une solution potentielle.

Ce mémoire est la suite logique du travail effectué lors du stage qui s'est déroulé durant le premier quadrimestre de l'année académique 2017–2018 à l'Université du Luxembourg. Ce stage avait pour but de répondre à la problématique exposée ici, et dont la solution est exposée en détail à travers ce document.

La structure est la suivante. Le chapitre 1 décrit les 3 concepts et paradigmes abordés et utilisés dans le cadre du mémoire. Ensuite, le chapitre 2 expose la problématique et les questions de recherche générales, permettant ainsi de poser le cadre dans lequel la solution proposée dans ce mémoire se développe. Le chapitre 3 détaille cette solution en expliquant les choix qui ont été posés. Pour permettre une meilleure compréhension des différents aspects de cette solution, le chapitre 4 explore une preuve de concept grâce à un exemple concret. Enfin, la conclusion finalisera ce mémoire.

Chapitre 1

État de l'art

Dans ce chapitre, les différents concepts principaux abordés tout au long de ce document seront explorés, afin de faire l'état des connaissances existantes. Le premier concept sera le *contrôle d'accès* introduit à partir de la section 1.1. Ensuite, c'est tout ce qui concerne le *proactive computing* qui sera développé dans la section 1.2. Enfin, la section 1.3 aborde brièvement le paradigme de l'*Internet des Objets*.

1.1 Contrôle d'accès

1.1.1 But

Le contrôle d'accès joue un rôle important dans la sécurité des systèmes d'informations. Le but du contrôle d'accès est de limiter les actions ou les opérations que pourrait effectuer un utilisateur sur un système informatique [1], et donc limiter les utilisations non autorisées. Il peut ainsi s'agir de contrôle d'accès physique ou logique.

Comme son nom l'indique, le contrôle d'accès physique concerne tout dispositif dont le but est de réguler l'accès à quelque chose de tangible physiquement. Cela peut être, par exemple, un bâtiment, une pièce ou un véhicule. Le contrôle d'accès logique, souvent couplé au contrôle d'accès physique, permet de limiter l'accès à un système d'information et tout ce qui le compose.

Le contrôle d'accès en général reprend quelques concepts fondamentaux [2] retrouvés dans une grande partie des modèles de contrôle d'accès :

1. Un sujet : l'entité qui veut accéder à l'objet protégé.
2. Un objet : constitue l'entité sur laquelle un contrôle d'accès est nécessaire.
3. Une action : ce que le sujet souhaite faire par rapport à l'objet (lire, modifier, etc.).
4. Un moniteur de référence : entité qui joue le rôle de médiateur entre les sujets et les objets.

Notons que le contrôle d'accès coexiste avec d'autres services proposés par un système de sécurité, comme l'authentification et l'audit. Il est important de faire une distinction entre le contrôle d'accès et l'authentification.

D'une part, le contrôle d'accès concerne l'autorisation d'un accès à une certaine ressource (objets), où la demande d'accès est faite par un utilisateur légitime (i.e. authentifié). L'authentification peut donc être vu comme un prérequis au contrôle d'accès [1].

D'autre part, l'authentification constitue le processus permettant de vérifier correctement l'identité d'un sujet. Ce processus est souvent basé sur un ou plusieurs facteurs [3] : quelque chose que nous connaissons (comme un mot de passe), quelque chose que nous possédons (une carte de sécurité par exemple), quelque chose que nous sommes (telle qu'une empreinte digitale) ou l'endroit où nous nous trouvons (via une localisation GPS par exemple). Ces facteurs peuvent aussi être combinés en une authentification multi-facteurs, comme par exemple avec l'usage d'une carte bancaire.

1.1.2 Politiques, modèles et mécanismes

Comme décrit dans [4], un système de contrôle d'accès doit prendre en compte trois abstractions : les politiques de contrôle d'accès, les modèles et les mécanismes.

Les politiques sont des exigences ou des règles de haut-niveau permettant de définir la façon de gérer les accès et qui peut, dans un certain contexte, accéder à quelles informations. Une politique de sécurité peut être spécifique à une seule application, ou au contraire s'appliquer dans un contexte multi-organisationnel. Ainsi, une politique peut concerner l'usage d'une ressource ou se baser sur des facteurs comme l'autorité, les compétences, les obligations, etc.

C'est lorsqu'il faut appliquer une politique qu'un mécanisme entre en action. Celui-ci va s'assurer que la requête d'accès faite par un sujet est bien interceptée et traduite pour qu'il soit possible d'y opposer une politique. Il doit fonctionner comme un moniteur de référence possédant les propriétés suivantes [5] :

— infalsifiable (*tamper-proof*) : il ne devrait pas être possible de le modifier.

— incontournable (*non-bypassable*) : c'est par là que doit transiter tous les accès au système et à ses ressources.

— confiné (*security kernel*) : les fonctionnalités de sécurité doivent être confinées dans une partie limitée du système.

— petit (*small*) : il ne doit pas être de taille trop conséquente, afin de permettre une vérification rigoureuse.

Un mécanisme est une fonction d'assez bas niveau et pour éviter de décrire ou d'analyser un système de contrôle d'accès uniquement à ce niveau, les modèles permettent de formaliser la représentation de la politique de sécurité imposée par un système et comment elle fonctionne. Un modèle permet en quelque sorte de faire le lien entre politique et mécanisme, entre la définition et l'implémentation. Il permet aussi de démontrer les propriétés de sécurité dont un système appliquant le modèle pourra bénéficier.

1.1.3 Différents modèles de contrôle d'accès

Les modèles de contrôle d'accès sont généralement divisés en deux catégories : discrétionnaire et non-discrétionnaire. Un modèle discrétionnaire est généralement associé à un contrôle d'accès basé sur l'identité des sujets. Un modèle non-discrétionnaire concerne les contrôles basés sur des règles.

Les plus connus sont les modèles *Discretionary Access Control* (DAC), *Mandatory Access Control* (MAC) et *Role-based Access Control* (RBAC). Ceux-ci seront introduits respectivement dans les sections 1.1.3.1, 1.1.3.2 et 1.1.3.3. Mais le modèle centrale qui est abordé plus en détail à travers la suite de ce document, c'est le modèle *Attribute-based Access Control* (ABAC), introduit à la section 1.1.3.4.

1.1.3.1 Discretionary Access Control

La politique DAC se base sur l'identité du demandeur d'accès pour vérifier si oui ou non un accès peut être autorisé. Elle est souvent utilisée pour définir ce qui peut être fait sur un objet, pour en limiter l'accès. Le propriétaire de l'objet, qui est une notion intrinsèque à ce modèle, peut définir qui peut faire quoi sur les objets qu'il contrôle. Un propriétaire peut aussi donner des droits d'accès à d'autres utilisateurs. Cette politique est très utilisée, notamment dans la plupart des systèmes d'exploitation.

Un modèle très répandu pour DAC est la *matrice de contrôle d'accès*. L'état du système de contrôle d'accès est défini par un triplet (S, O, A) [1], où S représente les sujets qui peuvent utiliser leurs privilèges, O représente les objets sur lesquels des privilèges s'appliquent et A représente la matrice de contrôle d'accès. Les lignes de celle-ci correspondent aux sujets et les colonnes correspondent aux objets. Ainsi, une entrée $A[s, o]$ ($s \in S$ et $o \in O$) donne les privilèges de s sur o. Un exemple d'une telle matrice est disponible à la figure 1.1.

	ressource 1	**ressource 2**	**ressource 3**	...	**ressource** n
sujet 1	rwx	rwx	rwx	...	rwx
sujet 2		r	r	...	rwx
sujet 3	rw	r	r	...	rwx
...					
sujet n	rw	rw	rw	...	rwx

FIGURE 1.1 – Exemple d'une matrice de contrôle d'accès.

Un système n'utilise généralement pas directement une matrice pour sauvegarder les droits de contrôle d'accès, car une telle matrice peut vite s'avérer être compliquée à gérer. Ainsi, c'est soit la méthode utilisant une liste de contrôle d'accès (*ACL, i.e. Access Control List*), soit l'approche *Capability* qui est utilisée. Dans l'approche *ACL*, la matrice est enregistrée par colonne. Chaque objet est associé à une liste indiquant pour chaque sujet quelle(s) action(s) il peut effectuer sur l'objet en question. Dans l'approche *Capability*, ce sont les lignes qui sont utilisées pour le stockage. Chaque sujet se voit attribuer une liste reprenant les objets et les droits d'accès possibles pour chacun d'eux.

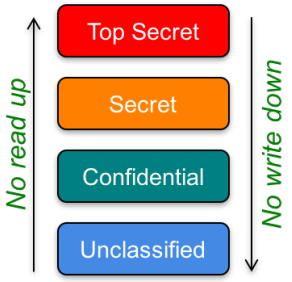

Figure 1.2 – Propriétés du modèle de Bell et La Padula.
Source: `https://www.cs.rutgers.edu/~pxk/419/notes/access.html`

Le principal désavantage du modèle DAC réside dans le fait qu'autoriser les utilisateurs à contrôler les permissions sur les objets qu'ils possèdent, rend le modèle vulnérable à une attaque de type « cheval de Troie ». De plus, il est difficile de maintenir un système DAC ainsi que de l'auditer, puisque les objets et les sujets peuvent être très nombreux.

1.1.3.2 Mandatory Access Control

Mandatory Access Control est la plus conséquente des politiques non-discrétionnaires. Dans ce type de politique, les droits d'accès sont régulés par une autorité centrale. Ainsi, l'utilisateur n'a aucun contrôle sur la politique d'accès, il ne peut donc pas changer les droits sur un objet qu'il possède. Cela permet entre autres de pouvoir mieux contrôler le flux d'information, mais aussi l'intégrité et la confidentialité de l'information [3].

Traditionnellement, cette politique est liée à la sécurité multi-niveau (*MLS, i.e. Multilevel Security*), qui est généralement présent dans la sécurité militaire. Les sujets et les objets sont étiquetés à un niveau de sécurité donné. C'est donc le principe du « *Need-to-know* » qui est d'application et la division en plusieurs catégories peut se faire comme suit (dans un ordre décroissant de niveau de sécurité) : Top secret, Secret, Confidentiel et Non-classifié.

Dès lors, il est possible d'assurer la confidentialité en utilisant le modèle de Bell – La Padula. Ce dernier formule deux propriétés, illustrées à la figure 1.2, qu'un système doit rencontrer pour assurer la confidentialité des informations [5] :

no read-up Il ne doit pas être possible pour un sujet de lire une information possédant un niveau de sécurité *supérieur* à celui du sujet.

no write-down Il ne doit pas être possible pour un sujet de modifier (ou créer) une information possédant un niveau de sécurité *inférieur* à celui du sujet.

Le modèle de Bell – La Padula ne garantit pas l'intégrité des données. C'est pourquoi, il est nécessaire d'utiliser un autre modèle pour s'en assurer, comme celui de Biba. En contraste au modèle de Bell – La Padula, un système respectant ce modèle possède les deux propriétés suivantes [5] :

no read-down Un sujet à un niveau d'intégrité donné ne doit pas pouvoir lire une information à un niveau d'intégrité plus faible.

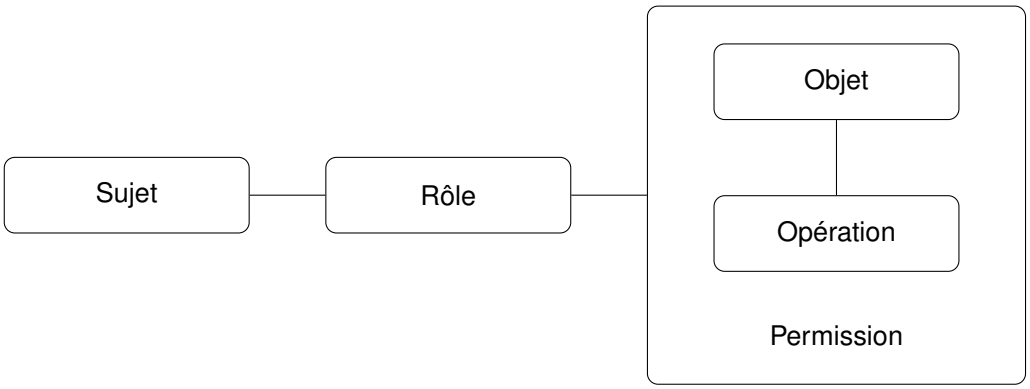

FIGURE 1.3 – Composants principaux du modèle RBAC.

no write-up Un sujet à un niveau d'intégrité donné ne doit pas pourvoir écrire une information à un niveau d'intégrité supérieur.

L'utilisation de la politique MAC est très répandue, notamment car cette politique fournit une meilleure sécurité que le modèle DAC. Elle permet entre autres de contrôler les flux d'informations indirect. Par exemple, elle est utilisée dans les système GNU/Linux comme avec le module SELinux, mais aussi dans certaine version du système d'exploitation Windows avec la fonctionnalité *Mandatory Integrity Control* introduite à partir Windows Vista. Mais elle présente aussi des désavantages, comme le fait qu'elle peut devenir trop rigide [2] ou qu'elle peut être laborieuse à implémenter.

1.1.3.3 Role-based Access Control

Le modèle *Role-based Access Control (RBAC)* a été introduit par Ferraiolo et Kuhn en 1992 et a été approuvé comme standard ANSI en 2004 [3]. Il a été développé pour remédier aux désavantages des politiques DAC et MAC, mais aussi pour faciliter l'administration des permissions pour un grand nombre d'utilisateurs.

La motivation principale est le besoin d'une politique de sécurité basée sur les sujets, afin d'évaluer les accès en fonction des compétences, ou en fonction du concept du moindre privilège [6]. Cette politique devrait être appliquée tout en ne perturbant pas la structure organisationnelle. C'est pourquoi il est nécessaire de restreindre les accès en utilisant la fonction ou le rôle d'un utilisateur. Et comme les rôles sont généralement stables au sein d'une organisation, alors que les utilisateurs et les permissions peuvent changer fréquemment, la gestion et la vérification du contrôle d'accès se trouvent simplifiées. Cette vision vient aussi du constat qu'au sein d'une organisation, les utilisateurs finaux ne sont pratiquement jamais les propriétaires des informations qu'ils manipulent (et donc sur lesquelles ils ont un droit d'accès). C'est généralement l'organisation elle-même qui en est propriétaire, le contrôle d'accès doit donc se faire au niveau de la fonction qu'un employé occupe, et non plus au niveau de la propriété comme dans les modèles précédents.

Les composants importants de ce modèle sont repris à la figure 1.3. Il fonctionne de la façon suivante. Un utilisateur, qui est défini comme un être humain (même si cette notion peut être étendue aux machines, réseaux, etc.), peut être autorisé à

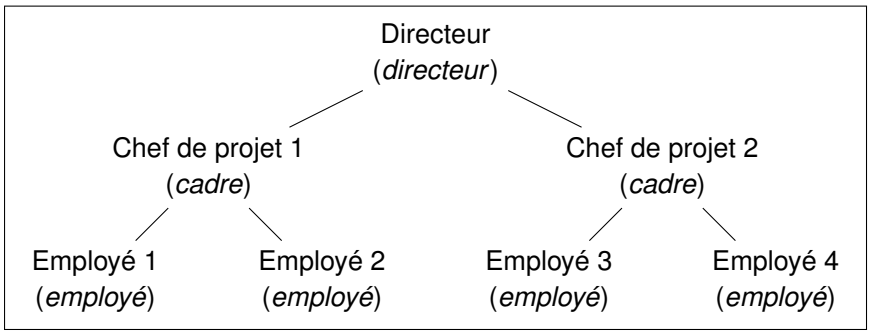

Figure 1.4 – Exemple d'une hiérarchie de rôles dans le modèle RBAC.

réaliser un ou plusieurs rôles. Un rôle est une fonction qu'un employé peut prendre dans le contexte d'une organisation et qui est donc associée à une certaine autorité et responsabilité. Les objets sont les ressources soumises au contrôle d'accès et une permission représente l'autorisation d'effectuer une opération sur un ou plusieurs objets. La nature d'une opération dépend principalement du type de système dont il est question. Par exemple, dans le cas d'un système de gestion de base donnée, il pourrait s'agir de l'insertion, modification, suppression de données. De plus, une ou plusieurs permissions sont assignées à des rôles spécifiques.

Il existe plusieurs bénéfices à l'utilisation du modèle RBAC [2] :

— Principe du moindre privilège : les rôles permettent de donner aux utilisateurs le plus petit privilège nécessaire pour effectuer une tâche. Ainsi, les utilisateurs liés à des rôles plus élevés n'ont besoin de les utiliser que lorsque cela est nécessaire.

— Séparation des rôles : aucun utilisateur ne devrait se voir assigner assez de privilèges pour utiliser le système de façon mal-intentionnée. Cette séparation peut être faite statiquement (les rôles assignés à un utilisateur sont fixes) ou dynamiquement (les rôles dépendent de la tâches courantes). La séparation dynamique est lié au concept de session, qui est une relation entre un utilisateur et un ou plusieurs rôles, activés pour la session courante.

— Application de contraintes : par exemple, il est possible de restreindre le nombre maximum d'utilisateur pour un rôle.

Un autre aspect clé est la hiérarchie de rôles qui permet de trier les rôles de manière plus naturelle en reflétant la ligne d'autorité et de responsabilité au sein d'une organisation. L'héritage se définit en termes de permissions, c'est-à-dire si le rôle r_2 hérite du rôle r_1, tous les privilèges de r_1 sont aussi des privilèges de r_2. Par exemple, chaque employé d'une entreprise possède le rôle « *employé* ». Le chef de projet possède le rôle « *cadre* », qui hérite de toutes les permissions du rôle « *employé* ». Enfin, au-dessus le directeur détient le rôle « *directeur* » qui hérite aussi des permissions du rôle « *cadre* ». Un exemple d'illustration en arbre de cette hiérarchie se trouve à la figure 1.4 (la propagation des permissions se fait donc du bas vers le haut).

RBAC possède tout de même un désavantage dans certains contextes : « l'explosion » des rôles. Dans une organisation, le nombre de rôles différents peut devenir

très conséquent et gérer tous ces rôles se complexifie au fil du temps puisqu'il faudrait idéalement les trier en hiérarchie, mais en pratique, la construire devient difficile et coûteuse [7].

1.1.3.4 Attribute-Based Access Control

Le dernier modèle de contrôle d'accès décrit dans ce chapitre s'intitule *Attribute-based access control* (ABAC). Son but principal est de fournir une grande flexibilité dans la façon dont le contrôle d'accès doit être appliqué, ce qui n'était pas possible dans les autres modèles puisque ceux-ci se basent traditionnellement sur l'identité de la personne qui essaie d'accéder à une ressource protégée. Ainsi, le modèle ABAC considère que le contrôle d'accès ne concerne pas toujours l'identité du sujet ou les rôles qui lui sont liés. En effet, des informations additionnelles peuvent influencer une décision, comme l'état courant du sujet, de ses actions et de l'environnement. Dans le cas de ABAC, une décision par rapport à une demande d'accès est donc basée sur certaines propriétés (appelée *attributs*). Il diffère donc des autres modèles en replaçant les sujets par un ensemble d'attributs et en remplaçant les objets par des descriptions, en termes de propriétés associées avec ceux-ci [2].

Un sujet est une entité associée à des attributs qui définissent son identité et ses caractéristiques. La nature du sujet peut être variée, il peut s'agir d'un utilisateur (personne humaine), d'un processus logiciel, un programme dans un environnement réseau, etc. Les attributs peuvent aussi être très variés puisqu'il peut théoriquement s'agir de n'importe quelle information qu'il est possible de connaître sur un sujet. Cela peut donc être le nom, l'organisation, la place occupée au sein de l'entreprise ou encore le rôle du sujet. De plus, comme dit ci-avant, un attribut ne se concentre pas uniquement sur le sujet, mais peut aussi s'axer sur l'objet ou sur l'environnement. L'heure, la position géographique, le type de ressource (dossiers médicaux, comptes en banque), l'action effectuée (lire, écrire, supprimer) sont des exemples d'attributs qu'il est possible de prendre en compte. Ceux-ci ne concernent donc pas nécessairement les sujets, mais ils peuvent néanmoins se révéler intéressant à la prise de décision.

Les attributs peuvent être combinés dans une ou plusieurs politiques (aussi appelées règles), qui seront évaluées lorsque c'est nécessaire et une réponse (autoriser ou non l'accès) sera renvoyée en fonction du résultat de l'évaluation. Cela permet de fournir un grand nombre de combinaisons possibles au sein des politiques, qui seront limitées uniquement par le langage de programmation et la richesse des attributs disponibles [8]. Il y a aussi un aspect dynamique, une décision peut varier si les valeurs des attributs varient, sans qu'il soit nécessaire de modifier les relations entre sujets et objets définies dans les règles.

Un scénario basique dans le cadre de ce modèle est le suivant [9] :

1. Un sujet demande l'accès à un objet.
2. Le mécanisme de contrôle d'accès évalue, pour prendre une décision, (a) les règles, les attributs liés (b) au sujet, (c) à l'objet et (d) à l'environnement.
3. Si l'accès est autorisé, le sujet accède à l'objet.

ABAC propose principalement une idée pour un nouveau modèle plus flexible de contrôle d'accès. Il n'existe pas de consensus sur les fonctionnalités exactes que ce modèle fournit, au-delà du schéma basique d'association d'attributs avec des sujets, des objets et de l'environnement, il n'y a que peu de consistance entre les implémentations d'ABAC [8]. Toutefois, un framework assez connu respectant les caractéristiques du modèle ABAC est le standard XACML, décrit en détails à la section 1.1.4.

Avantages principaux du modèle ABAC

Dans les modèles décrits précédemment, le contrôle d'accès était principalement basé sur l'identité du sujet qui souhaitait réaliser une action sur une ressource. Cette approche peut être laborieuse à gérer puisque les accès aux objets par des sujets qui ne font pas directement partie de l'organisation contrôlant ces objets, requerrait que l'identité du sujet soit d'abord fournie pour être insérée ensuite dans une liste d'accès. De plus, les identités et les rôles qui qualifient un sujet sont souvent insuffisants dans l'expression des besoins de systèmes de contrôle d'accès réels. Par exemple, RBAC fonctionne en associant un sujet avec un rôle. Cela ne permet pas de supporter facilement des décisions à plusieurs facteurs, par exemple en prenant en compte la localisation du sujet. D'autant plus que les rôles sont basés sur un aspect statique de l'organisation, ce qui peut poser problème dans le cadre d'architecture RBAC où des décisions de contrôle d'accès doivent être dynamiques. [9]

C'est là qu'ABAC permet d'avoir un modèle de contrôle d'accès où les décisions de contrôle d'accès peuvent être prises sans connaissance préalable de l'objet par le sujet ou connaissance du sujet par le propriétaire de l'objet. En effet, le modèle ABAC permet de se baser sur le concept d'attributs des sujets et des objets, et ainsi de ne pas être obligé d'assigner des autorisations directement aux sujets individuels avant que ces derniers puissent réaliser des opérations des objets. Ceci permet notamment une plus grande flexibilité dans les grandes organisations, où gérer les listes de contrôle d'accès et les rôles peut devenir une tâche coûteuse en temps et complexe. Enfin, ABAC permet aussi d'implémenter les systèmes existants basés sur des rôles si nécessaire, ou le modèle peut aussi supporter la migration de tels systèmes vers des systèmes utilisant des politiques plus granulaires basées sur les différentes caractéristiques des sujets. [9]

Désavantages du modèle ABAC

Le principal désavantage du modèle ABAC réside dans ce qui fait aussi sa force : les attributs. En effet, tout est attribut dans ce modèle, ce qui permet une grande flexibilité comme nous l'avons vu précédemment, mais qui peut aussi engendrer une explosion des attributs. Cela montre que le modèle ABAC propose une sémantique de base assez pauvre, mais permettant une flexibilité unique à ce modèle.

De plus, le modèle ABAC complexifie l'architecture à mettre en place pour l'utiliser dans des systèmes réels. Les composants du modèle peuvent être disséminés à travers toute une organisation, et parfois sur des réseaux différents. Un simple accès à un document peut nécessiter l'évaluation d'une politique dont les attributs peuvent provenir de sources dispersées logiquement et physiquement. Les questions

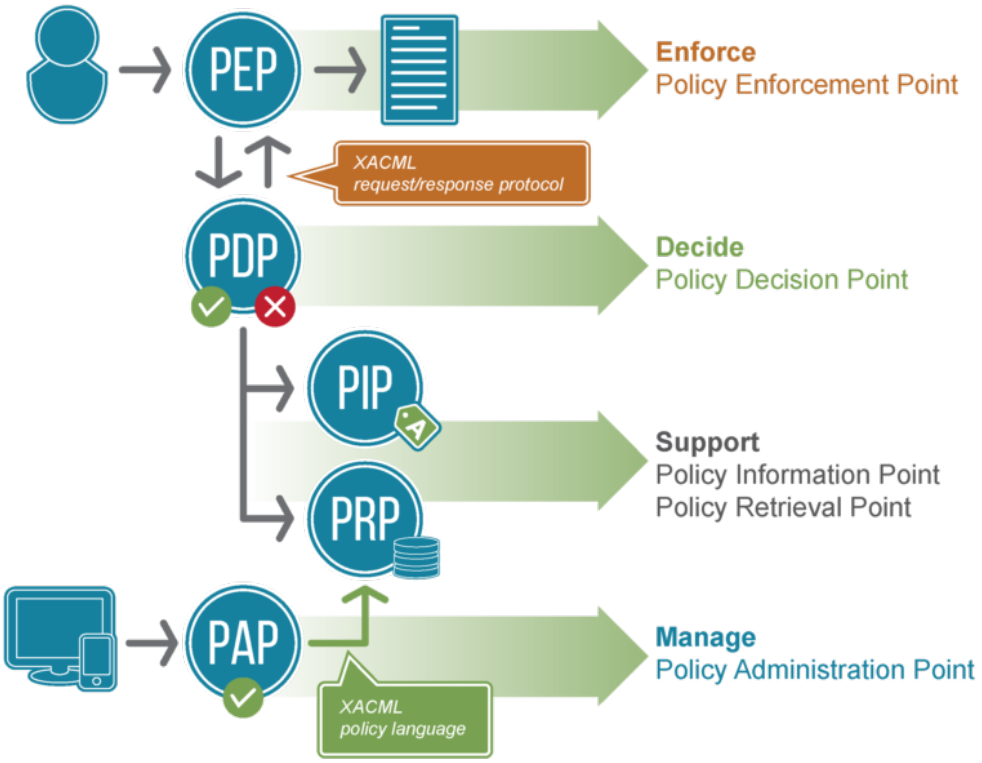

FIGURE 1.5 – Diagramme de l'architecture XACML.
Source: `https://www.axiomatics.com/100-pure-xacml`

de faisabilité et de performance sont donc à considérer lors du déploiement d'un système de sécurité basé sur le modèle ABAC. [9]

1.1.4 XACML

XACML est un acronyme pour *eXtensible Access Control Markup*. Il s'agit d'un standard défini par le consortium *OASIS*, dont l'un des objectifs principaux est de fournir une terminologie commune entre les implémentations de systèmes de contrôle d'accès basés sur ABAC. XACML formalise la définition d'un langage permettant de définir des politiques, un protocole pour les requêtes et les réponses, et une architecture pour évaluer les requêtes en fonction d'un ensemble de politiques. Plusieurs versions du standard ont été proposées, dont la dernière en date, la 3.0, qui a été standardisée en 2013. Le reste de cette section se base sur la spécification officielle [10].

1.1.4.1 Une architecture

L'architecture est composée 5 composants principaux [10], la figure 1.5 représente comment ils sont agencés entre eux.

— PEP (*Policy Enforcement Point*) : Ce nœud a pour rôle de « protéger » la ressource ou le service sur lequel s'applique le contrôle d'accès. Il est donc chargé d'intercepter les demandes d'accès faites par un sujet. La forme d'un

PEP peut être très variable en fonction du système dont il est question. Par exemple, il peut s'agir d'un composant au sein d'un serveur Web ou une partie d'un gateway dans un réseau.

— PDP (*Policy Decision Point*) : Il constitue le nœud central de l'architecture, c'est à lui que revient la tâche de prendre une décision finale. Lorsque que le PDP reçoit une requête de la part d'un PEP, il récupère les politiques applicables au contexte courant, pour ensuite les évaluer et approuver au non l'accès. Pour ce faire, le PDP s'appuie sur le PIP et le PDP.

— PRP (*Policy Retrieval Point*) : Son but est de stocker les politiques prédéfinies et de les rendre disponibles au PDP. Il s'agit typiquement d'une base de données ou un endroit du système de fichiers.

— PAP (*Policy Administration Point*) : Il est directement lié au PRP, puisqu'il sert d'interface permettant de gérer les politiques. Par exemple, il peut s'agir d'une interface homme-machine permettant à un administrateur de modifier, créer, supprimer, etc. les politiques.

— PIP (*Policy Information Point*) : Ce dernier nœud permet de fournir des informations complémentaires au PDP lorsque c'est nécessaire. En effet, il est possible que certaines politiques fassent référence à des attributs dont la valeur ne peut être trouvée dans le contexte directe de la requête. Les sources d'attributs derrière le PIP peuvent être variées. Cela peut aller de la base de données relationnelle à un fichier CSV, en passant par un répertoire LDAP.

Un dernier composant qui permet de faire la passerelle entre le monde « applicatif » et le monde « XACML », est appelé le *Context handler*. Il doit prendre en charge la tâche de conversion de la requête « brute » (spécifique au domaine d'application) vers un format canonique tel qu'expliqué à la section 1.1.4.3, pour ensuite la communiquer au PDP. C'est aussi lui qui devra prendre en charge la conversion entre les représentations d'attribut dans l'environnement applicatif et les représentations d'attribut dans le contexte XACML.

La réponse finale fournie par le PDP au PEP peut prendre une des quatre formes suivantes.

— *Permit* : L'accès demandé est autorisé.

— *Deny* : L'accès demandé est refusé.

— *Indeterminate* : Le PDP ne peut pas déterminé si l'accès est autorisé ou refusé car une erreur s'est produite lors de la phase d'évaluation.

— *NotApplicable* : Aucune politique applicable à la requête d'accès courante n'a été trouvée.

De plus, cette réponse finale peut aussi contenir deux composants facultatifs. Le premier est un ensemble d'*obligations*. Une *obligation* est une opération qui doit être obligatoirement exécutée par le PEP lorsqu'il reçoit la réponse du PDP. Le second est un ensemble d'*advices*. Un *advice* est une information supplémentaire qui peut être retournée en conjonction de la réponse.

1.1.4.2 Un langage

Le modèle du langage défini par le standard XACML, rendu concret dans des systèmes réels en utilisant la technologie XML, est décomposé en plusieurs composants, afin de fournir le plus de modularité possible. Ceux-ci sont illustrés à la figure 1.6 et expliqués dans les paragraphes suivants.

Les composants principaux sont structurés sur 3 niveaux. Le plus général est appelé *PolicySet*, il représente un ensemble de politiques. Une politique (*Policy*) se compose d'une ou plusieurs règles. Une règle (*Rule*) est l'élément le plus élémentaire d'une politique XACML.

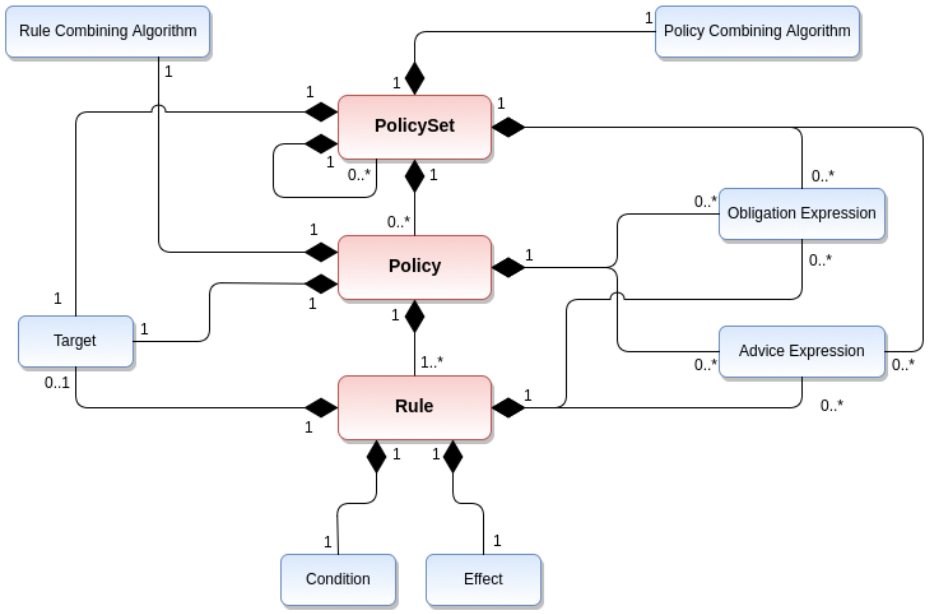

FIGURE 1.6 – Modèle du langage proposé par XACML.

L'élément *Target* peut se retrouver dans un *PolicySet*, une *Policy* ou une *Rule*. Dans tous les cas, il permet de définir un ensemble de requêtes auxquelles l'élément s'applique, ce qui permettra au PDP de connaître quels éléments récupérer lorsqu'il reçoit une requête. Il prend la forme d'une expression logique définie sur les attributs de la requête. Si l'élément *Target* est absent d'une règle, alors celle-ci hérite du *Target* de la *Policy* parente ou du *PolicySet* parent. Une règle possède aussi un élément *Effect* qui permet d'indiquer le résultat en cas de « bonne » évaluation de la règle. Ce résultat peut être soit « *Permit* », soit « *Deny* ». Enfin, l'élément *Condition* d'une règle permet de spécifier une expression booléenne qui vient raffiner le champ d'application de cette dernière (en plus du champ déjà défini par l'élément *Target*).

Deux éléments sont communs aux 3 composants principaux. L'un est l'*Obligation expression* et l'autre est l'*Advice expression*, qui remplissent les mêmes rôles que ceux décrits à la section précédente (1.1.4.1).

Les derniers éléments importants sont les algorithmes de combinaison, soit pour des politiques (*Policy Combining Algorithm*), soit pour des règles (*Rule Combining*

Algorithm). Un algorithme de combinaison pour les règles permet d'arriver à une décision en fonction de différents résultats d'un ensemble de règles. De façon similaire, un algorithme de combinaison pour les politiques permet d'arriver à une décision en fonction de différents résultats pour un ensemble de politiques. XACML propose plusieurs algorithmes pré-définis [10].

1.1.4.3 Un protocole requête/réponse

Le protocole défini par le standard XACML formalise les échanges de requêtes et de réponses entre le PEP et le PDP. Comme pour les politiques, c'est la technologie XML qui est utilisée. Cela permet principalement d'éviter de travailler directement sur des inputs et des outputs spécifiques au domaine d'application, qui peut varier fortement d'un environnement à l'autre, mais d'isoler le « travail » nécessaire au contrôle d'accès au sein d'un contexte XACML.

1.2 Proactive computing

L'informatique telle que beaucoup de personne la connaisse se veut résolument interactive. En effet, un système informatique possède la plupart du temps une interface permettant à un utilisateur d'interagir avec celui-ci. Les moyens d'interaction ne manque pas : écran, clavier, souris, écran tactile, etc. De plus, l'industrie de l'informatique n'a cessé d'évoluer, au cours des dernière décennies, vers des ordinateurs de plus en plus petits et de plus en plus présents. Ainsi, le nombre d'appareils informatiques qui nous entoure a dépassé notre capacité d'interaction avec ceux-ci. Il semble utopique de vouloir gérer et interagir avec ceux-ci tout en exploitant leur plein potentiel, surtout dans une ère où les systèmes « Internet des Objets » sont de plus en plus prépondérants. Cependant, une machine peut fonctionner en continu et sans interruption, hormis durant les possibles périodes de maintenance. Il y a donc un réel besoin d'une technologie permettant d'exploiter ce « manque ».

J. C. R. Licklider en 1960 avait déjà anticipé ce besoin. En effet, dans un de ses articles, sa vision était que l'homme et les machines pourraient travailler ensemble pour réaliser des tâches complexes. Il ne voyait donc pas les ordinateurs remplacer les êtres humains, mais il croyait en une symbiose, à l'image de celles que nous trouvons dans la nature, entre certains insectes et certains arbres par exemple. Selon lui, les hommes pourraient définir les buts, formuler les hypothèses, déterminer les critères et accomplir les évaluations. Quant aux machines, elles réaliseront le travaille routinier qui doit être fait afin préparer le terrain pour les idées et les décisions, dans un contexte technique et scientifique. [11]

Aux alentours des années 1990, Mark Weiser utilise le terme *ubiquitous computing* lorsqu'il expose sa vision du futur. Il explique que dans les années à venir, l'informatique deviendrait de plus en plus présente dans nos vies, jusqu'à s'y « fondre », sans que nous nous rendions compte de la présence des outils informatiques qui nous entourent. C'est aussi durant ces années que les premiers systèmes distribués font leur apparition. Ce sont tous ces concepts qui ont mené à l'apparition du *proactive computing*. Le terme « proactif » peut se diviser en deux mots, *pro* et *actif*. Le mot *pro* signifie « avant », « à la place de ». Et le mot *actif*, dans ce contexte, signifie « agir » ou « prendre des actions ». « Proactif » peut donc se définir comme le fait

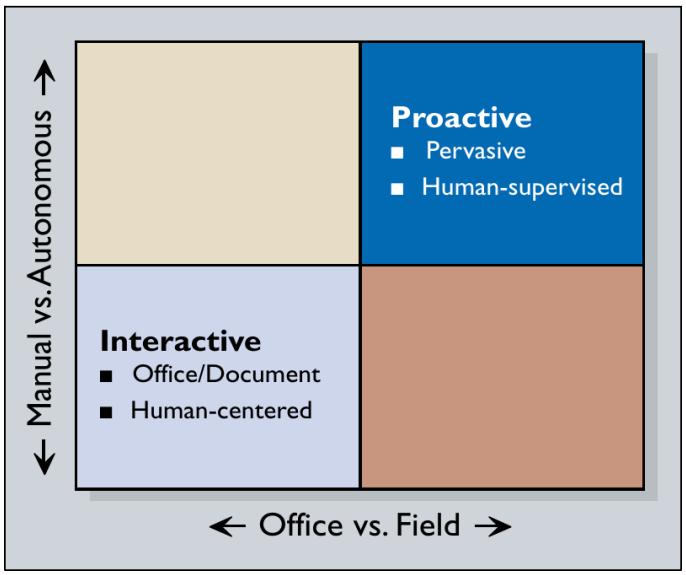

FIGURE 1.7 – Les 4 quadrants de l'informatique ubiquitaire.
Source: [12]

d'agir en anticipant les événements futurs, de prendre des décisions en anticipant. [12]

David Tennenhouse a introduit ce concept dans un article publié en 2000. Dans cet article, il définit le proactive computing comme un mouvement de l'informatique centrée sur l'homme vers l'informatique supervisée par l'homme, ou même non supervisée. Comme le montre la figure 1.7, l'informatique ubiquitaire de Weiser peut être divisé en deux axes. Un pour mesurer à quel point les applications sont centrées sur la bureautique. L'autre est en rapport avec le degré d'intérêt que les ordinateurs et leurs interfaces mettent, soit sur l'interaction avec les êtres humains, soit sur l'interaction avec le reste de l'environnement. David Tennenhouse constate que peu de chercheurs ont « cassé » avec la tradition, axée principalement sur l'interaction homme-machine et que les recherches se sont donc cantonnées dans le quadrant inférieur gauche de la figure 1.7. Le proactive computing représente un moyen d'explorer le quadrant supérieur droit. [12]

À ce moment, au début des années 2000, deux visions voient le jour : l'une est élaborée par IBM, l'*Autonomic computing*, qui est expliqué à la section 1.2.1.1. L'autre vient de chez Intel, où David Tennenhouse élabore alors le *Proactive computing*, détaillé à la section 1.2.1.2.

1.2.1 Deux visions

La figure 1.8 montre qu'il y a, entre les deux visions que sont l'*Autonomic computing* et le *Proactive computing*, un chevauchement intellectuel. En effet, elles sont nécessaires pour nous aider à passer de l'informatique interactive vers un nouveau mode de pensée, afin d'être prêt à gérer une situation où le rapport entre personne et processeurs serait de un pour des milliers, et où le flux de données qu'ils fourniraient serait plus que conséquent. [13]

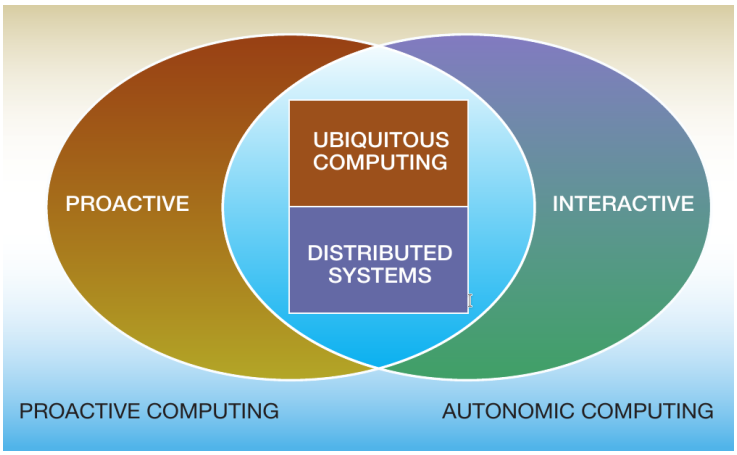

Figure 1.8 – Relation entre l'Autonomic computing et le Proactive computing. *Source*: [13]

1.2.1.1 Autonomic computing

C'est en 2001, avec le constat que pour continuer à avancer dans l'industrie informatique, il y a un obstacle majeur à franchir : une crise dans la complexité des systèmes informatiques, approchant à grands pas [14]. Les systèmes et leurs environnements peuvent s'étendre sur des millions de lignes de codes, et installer, configurer, maintenir, etc. ces systèmes requiert beaucoup de main d'œuvre et des personnes spécialisées. La complexité des systèmes semble atteindre les limites des capacités humaines, ce qui rend les architectes de moins en moins aptes à anticiper et concevoir les interactions entre les différents composants.

Ainsi, un chercheur nommé Paul Horn, a introduit l'idée de l'Autonomic computing. L'idée principale est d'arriver à des systèmes capables de s'« autogérer », en fonction d'objectifs hauts-niveaux. Le terme *Autonomic* n'a pas été choisi au hasard, sa connotation biologique étant souhaitée, afin de rappeler comment notre système nerveux s'occupe de réguler notre battement de cœur ou la température de notre corps, permettant à la partie consciente de notre cerveau de ne pas devoir y penser constamment et donc de se concentrer sur d'autres tâches. Ce terme, d'un point de vue informatique, reflète l'idée de systèmes capables de se contrôler « naturellement » par soi-même. L'intention est de libérer les administrateurs systèmes des détails de fonctionnement et de maintenance du système, mais également de fournir des machines capables de tourner 24h/24h aux meilleures performances possibles. C'est pourquoi, cette auto-gestion devra permettre aux systèmes de maintenir et ajuster leur fonctionnement en réponse à un changement au niveau de la charge de travail, de la demande ou d'autres conditions externes, mais aussi en cas de défaillance logicielle ou matérielle. IBM a défini 4 aspects essentiels à l'auto-gestion [14], décrits ci-après.

Self-configuration La plupart des sites Web de grande taille ou des data centers sont composés de beaucoup de composants différents : serveurs, routeurs, base de données et autres technologie pouvant venir de vendeurs différents. Lorsqu'il faut implémenter un nouveau composant dans une structure existante,

cela peut prendre beaucoup de temps et d'experts, sans parler des coûts pour l'organisation. Dans le cas d'un système s'auto-gérant, il sera capable de se configurer automatiquement en respectant certaines politiques de haut-niveau spécifiant ce qui est désiré et non pas ce qui doit être fait pour y arriver. Un nouveau composant pourra donc s'intégrer de manière homogène et transparente, sans que le système existant ne doive être interrompu, lui permettant de s'adapter à la présence du nouveau venu. [14]

Self-optimization Les gros middlewares informatiques peuvent souvent être paramétrés par un nombre important de paramètres qui doivent être défini correctement, afin de permettre au système de fonctionner optimalement. Cependant, trouver les bonnes valeurs pour ces paramètres s'avère compliqué dans la plupart des cas. C'est là qu'un système autonome pourrait continuellement chercher le bon paramétrage d'un composant, dans le but d'arriver à un fonctionnement optimal et efficace, aussi bien en termes de performance qu'en termes de coût. Pour ce faire, le système surveillera quels effets sont engendrés par une modification d'un paramètre et apprendra à faire les bon choix au fur et à mesure. [14]

Self-healing Dans les grandes organisations de l'industrie informatique, une partie conséquente du personnel peut être amenée à identifier, tracer et déterminer la cause principale d'une défaillance dans des systèmes complexes. Cela peut durer des semaines avant que la cause d'un problème soit diagnostiquée et réparée. Un système autonome doit être capable de gérer ces problèmes. Il devra donc détecter le problème, identifier la cause en analysant les différents informations à sa disposition (fichiers de log ou toutes autres données de monitoring) et réparer en appliquant des patchs logiciels pré-établis ou alerter un administrateur. [14]

Self-protection La plupart des systèmes existants sont déjà protégés grâce à des firewalls et des outils de détection d'intrusion, mais le choix de la protection à appliquer pour contrer les potentielles attaques, revient tout de même au programmeur ou à l'administrateur d'un système. Les systèmes autonomes pourront se protéger de deux façons. D'un côté, ils protégeront le système dans l'ensemble, notamment contre les attaques à grande échelle pouvant créer des problèmes que la capacité de *self-healing* pourrait ne pas être en mesure de réparer. D'un autre côté, ils pourront anticiper un problème en se basant sur les informations disponibles à travers des senseurs, et ainsi éviter ou limiter les dégâts. [14]

Enfin, d'un point de vue architectural, un système autonome devrait être structuré comme une collection d'éléments autonomes interagissant entre eux. Un élément autonome se définit comme un constituant du système qui contient des ressources et qui fournit des services à des humains ou à d'autres éléments autonomes. Ces éléments autonomes s'occuperont de gérer leur comportement interne et leurs relations avec d'autres éléments, en concordance avec les politiques définies par les humains ou par d'autres éléments. Ils auront aussi un cycle de vie complexe, s'occupant de plusieurs fils d'activité et réagissant continuellement aux stimuli de leur

environnement. De tels éléments auront donc, comme caractéristiques principales, l'autonomie, la proactivité et l'interactivité [14].

1.2.1.2 Proactive computing

C'est dans son article paru en 2000 que David Tennenhouse parle pour la première fois du proactive computing. Ce concept peut être apparenté à l'Autonomic computing décrit ci-avant, mais le proactive computing cherche aussi à explorer d'autres domaines d'applications où il pourrait être utile (et non pas seulement la domaine de l'auto-gestion). L'idée principale est permettre la transition à partir des systèmes interactifs tels que nous les connaissons, vers des environnements proactifs qui anticipent nos besoins et agissent à notre place [13]. La conception de système proactif se base sur sept principes.

Connexion avec le monde physique Dans le modèle existant, la plupart des ordinateurs se connecte à travers un réseau vers des serveurs distants. Ce mode de fonctionnement nous permet d'agir sur de l'information qui pourra avoir un effet sur le monde physique, mais par le biais de personnes. Cependant, si nous voulons permettre à l'informatique de nous aider dans notre vie de tous les jours, il faut équiper le monde physique d'appareils de manière à ce que les systèmes informatiques puissent avoir une connaissance précise de l'état de l'environnement, pour ensuite pouvoir utiliser celle-ci afin d'agir sur ce dernier.

Fermer la boucle Pour pouvoir intégrer les systèmes avec le monde physique, une réponse en temps-réel est nécessaire. Mais l'informatique interactive place l'homme dans la boucle de contrôle, dans l'attente de ses instructions. Afin de réussir une intégration complète, les systèmes doivent répondre plus rapidement que lorsqu'une personne est impliquée dans la boucle de contrôle, ils doivent répondre en temps-réel aux événements du monde physique.

Mise en réseau Les sytèmes proactifs doivent être interconnectés afin de pouvoir communiquer avec le plus de systèmes possibles, et ainsi accéder à une grande quantité de données.

Macro traitement Cela permettra aux systèmes proactifs d'exécuter certaines tâches automatiquement.

Gérer l'incertitude Puisqu'un système proactif travaillera en conjonction avec le monde tangible, une certaine incertitude sera toujours présente. Il devra alors fonctionner avec cette contingence.

Anticipation Cette caractéristique est la plus importante pour vraiment faire preuve de proactivité, il faudra qu'un système proactif prédise en quelque sorte le futur. Se concentrer sur le contexte, sur des raisonnements statistiques et les données, peut servir de base de raisonnement pour anticiper les besoins d'un utilisateur.

Rendre les systèmes personnels Un système proactif doit pouvoir être adapté pour un utilisateur spécifique, et donc s'adapter à ses besoins.

David Tennenhouse a donc formulé là une idée prometteuse, mais afin de la rendre concrète, une mise en œuvre des principes énoncés était encore nécessaire. Ainsi, une implémentation a été réalisée à l'Université du Luxembourg ; elle est expliquée à la section suivante (1.2.2).

1.2.2 Implémentation existante d'un moteur proactif

Une implémentation d'un moteur proactif a été réalisée à l'Université du Luxembourg par une équipe de recherche sous la supervision du Professeur Zampunieris. C'est cette implémentation qui sera utilisée dans la suite de ce mémoire.

1.2.2.1 Contexte

Le moteur a été développé dans le cadre d'une recherche sur l'amélioration de l'apprentissage à l'Université du Luxembourg. Ce concept est appelé *Learning Management Systems (LMS)* ou plate-forme d'*e-learning*, les exemples les plus connus sont ATutor ou Moodle^{TM}. Dans [15], le moteur proactif est utilisé dans un nouveau type de LMS proactif, conçu pour aider les utilisateurs à mieux interagir en ligne avec un environnement éducatif, en fournissant une analyse programmable, automatique et continuelle des interactions des utilisateurs, en plus de diverses actions exécutées par le LMS lui-même. Un tel système agissait principalement en passant par la base de données existante de l'outil déjà en place (dans ce cas-ci Moodle^{TM}).

1.2.2.2 Règles et scénarios

Le moteur proactif est un *Rule-running system*, il exécute des règles pré-définies à une certaine fréquence. Une règle est l'élément élémentaire du moteur proactif. Elle se divise en cinq parties : « acquisition des données », « gardes d'activation », « conditions », « actions » et « génération de règle(s) ».

Acquisition des données (1) Cette première étape permet d'acquérir les données nécessaires à la bonne exécution de la règle. Ces données sont récupérées via le wrapper qui sert d'intermédiaire entre la base de donnée externe et le moteur. Elles sont ensuite stockées dans des variables locales de la règle, qui ne peuvent pas être modifiées lors du reste de l'exécution.

Gardes d'activation (2) La deuxième étape permet de vérifier que les conditions d'exécution de la règle sont bien rencontrées. Concrètement, il s'agit d'une expression booléenne comprenant plusieurs tests sur les variables locales initialisées à l'étape précédente. Cette étape agit comme un déclencheur pour les deux étapes suivantes, la troisième et la quatrième.

Conditions (3) Si la deuxième étape est validée, cette troisième étape peut être exécutée. Son but est sensiblement similaire à l'étape 2, mais elle permet d'aller plus loin dans les tests faits sur le contexte et s'assurer que la quatrième

étape peut s'exécuter. Elle permet notamment d'éviter de faire des tests qui ne sont peut-être pas utiles dans l'étape 2, et donc d'alléger cette dernière.

Actions (4) Si l'étape 2 et 3 ont été validées, l'étape 4 peut être réalisée. Celle-ci est le cœur de la règle, puisqu'elle contient ce pour quoi la règle existe. Elle fournit également beaucoup de liberté puisque, comme expliquée ci-après, une règle est implémentée en utilisant le langage de programmation Java. Le programmeur peut dès lors exploiter toutes les possibilités qu'offre le langage.

Génération de règle(s) (5) Enfin, la cinquième étape conclut l'exécution de la règle en permettant la création d'autres règles. La règle courante peut aussi choisir de se cloner afin de rester « en vie » dans le moteur proactif.

Comme le moteur est implémenté avec le langage Java, une règle est définie en tant qu'une classe Java. Le squelette d'une telle classe est disponible au listing 1.1, où chaque méthodes représente respectivement les étapes définies ci-avant.

Listing 1.1 – Squelette d'une règle définie en Java.

```java
public class Rule extends AbstractRule {

    // Définitions des variables locales
    // ...

    public Rule() { ... }

    @Override
    protected void dataAcquisition() { ... }

    @Override
    protected boolean activationGuards() { ... }

    @Override
    protected boolean conditions() { ... }

    @Override
    protected void actions() { ... }

    @Override
    protected boolean rulesGeneration() { ... }
}
```

Avec ce concept de règle, il est possible de créer des scénarios. Un scénario peut être vu comme une construction abstraite permettant de lier plusieurs règles ensemble. Cet ensemble de règles suit un certain « chemin » qui peut être composé de point de choix, la façon dont le scénario se déroule peut donc varier d'une exécution à l'autre, en fonction de l'état des données contextuelles. Ainsi, ce concept

a principalement pour but de rassembler les différentes règles qui sont nécessaires pour effectuer une tâche précise.

1.2.2.3 Fonctionnement interne

À chaque itération, le moteur proactif exécute plusieurs règles. Cette exécution est paramétrable grâce à trois paramètres, permettant de changer le comportement du moteur en fonction du contexte dans lequel il est utilisé :

— `F` : Définit la fréquence des itérations.

— `N` : Permet de limiter le nombre maximum de règles exécutables sur une itération.

— `P` : Correspond au temps minimum entre deux itérations. Il est utile uniquement lorsque le temps d'exécution d'une itération venait à être trop long. Par exemple, posons x comme le temps d'exécution. Si $T_{restant} = \texttt{F} - x$ est inférieur à `P`, alors le moteur devra être mis en pause pendant `P`, sinon il le sera pendant $T_{restant}$.

Une itération du moteur exécute donc plusieurs règles, et la façon dont se déroule l'exécution de l'une d'elles est définie au listing 1.2 [15].

Listing 1.2 – Algorithme principal du moteur proactif.

```
1.  Pour chaque demande d'accès DA
    a.  Exécuter DA
    b.  Si erreur alors lever une exception sur la console de gestion
        ↪ du système et aller à l'étape 7
        sinon créer une nouvelle variable locale et l'initialiser
            ↪ avec le résultat de DA
        créer une nouvelle variable locale booléenne "activated" à
            ↪ false

2.  Pour chaque garde d'activation GA
    a.  Évaluer GA
    b.  Si résultat == false alors aller à l'étape 6
        sinon si GA == dernier test de garde d'activation
              alors activated = true

3.  Pour chaque condition C
    a.  Évaluer C
    b.  Si résultat == false alors aller à l'étape 6

4.  Pour chaque action A
    a.  Exécuter A
    b.  Si erreur alors lever une exception sur la console de gestion
        ↪ du système et aller à l'étape 7

5.  Pour chaque génération de règle R
```

```
    a.  Exécuter R
    b.  insérer la nouvelle règle générée en tant que dernière règle
        ↪ du système

6.  Supprimer toutes les variables locales

7.  Supprimer la règle du système
```

Pour stocker les règles pendant la durée d'activité du moteur, celui-ci repose sur deux files basées sur le principe « first in, first out ». La première file, dénommée `currentQueue`, est utilisée comme file d'entrée d'une itération, elle contient les règles à exécuter. La seconde file, dénommée `nextQueue`, est utilisée comme file de sortie, elle sera remplie au fur et à mesure par les règles qui en ont créé d'autres (ou qui se sont clonées). À la fin de l'itération, la file `nextQueue` devient `currentQueue` et est vidée de son contenu pour être prêt à la prochaine itération.

1.2.2.4 Ressources

Le moteur proactif a été conçu pour fonctionner avec deux ressources (bien qu'il soit facile d'en rajouter), deux bases de données pour être précis.

Une est interne au fonctionnement du moteur, elle permet d'enregistrer différents paramètres, dont ceux décrits à la section précédente (1.2.2.3), mais aussi d'autres permettant d'agir sur l'exécution du moteur (utiles pour l'arrêt ou le redémarrage). Il est aussi possible d'y stocker les règles courantes à chaque fin d'itération (i.e. le contenu de la file `currentQueue`) et ainsi donner la possibilité de recommencer là où le moteur s'est arrêté en cas d'erreur par exemple.

La deuxième base de données est généralement celle du système sur lequel le moteur est greffé. C'est par là que les règles s'exécutant dans le moteur pourront agir sur le système hôte ou connaître son état. Nous pouvons noter que les paramètres `F` et `P` permettent de ne pas surcharger cette ressource externe, et ainsi laisser le système hôte faire son travail sans être trop ralenti par le moteur proactif. Il est donc important de bien paramétrer le moteur en fonction du système externe dont il est question.

1.2.3 Exemples de systèmes utilisant le proactive computing

1.2.3.1 PLMS – Proactive Learning Management System

Ce premier exemple a déjà été introduit à la section 1.2.2.1. Celui-ci avait pour but d'améliorer les capacités d'un *Learning Management System (LMS)* existant, *Moodle*^{TM}, en introduisant un comportement proactif dans son fonctionnement. Les scénarios du moteur, dans ce cas-ci, permettaient de prendre en charge certaines situations pré-définies comme des notifications, des rappels, prévention de problème, aide à l'utilisateur, etc. Ainsi, le processus d'apprentissage se voyait amélioré, notamment en permettant une approche individuel pour chaque étudiant [16].

1.2.3.2 PEMD – Proactive Engine for Mobile Devices

Ce deuxième exemple vise à fournir une solution pour améliorer les système mobiles existants en y ajoutant des propriétés proactives. Un tel système est appelé *Proactive Engine for Mobile Devices (PEMD)*, il permet entre autres la communication entre différents appareils pour échanger des informations [17].

Avec l'aide d'un PEMD, il a été possible de créer des applications telles que *SilentMeet*. Celle-ci est capable de détecter, en se basant sur la collaboration, les possibles réunions ou événements avec plus de 2 participants et de placer le smartphone en mode « silencieux » automatiquement, afin de minimiser les risques d'interruption. Pour cela, elle détecte et récupère certaines informations pertinentes (temps, localisation, événements dans l'agenda, etc.) à propos du contexte. Un utilisateur peut ainsi créer un événement auquel il ajoute des participants, qui doivent ensuite accepter de faire partie du groupe. Tous les appareils participants peuvent prendre des décisions globales grâce à la collaboration, qui permet de créer une connaissance générale partagée entre tous [18].

1.3 Internet des Objets

Le terme *Internet des Objets (IoD, ou IoT pour Internet of Things en anglais*[1]*)* est de plus en plus souvent mentionné depuis quelques années déjà. Certains le qualifient de « buzzword », alors que d'autre y voient l'avenir d'Internet, considéré comme le Web 3.0. Dans les faits, il estimé que l'IoT représentera plus ou moins 30 milliards d'objets connectés d'ici 2020 [19].

1.3.1 Un paradigme

Il existe plusieurs points de vue sur la manière de définir l'IoT, ce qui peut rendre difficile la compréhension exacte de ce paradigme et des idées qui y sont liées. Le terme *Internet des Objets* se compose de deux sous-termes, *Internet* et *Objets*. Le premier met l'accent sur une vision « réseau » du paradigme (*"Internet"-oriented*), alors que le deuxième s'intéresse à l'aspect « objets » (*"Things"-oriented*). Ces deux visions sont représentées à la figure 1.9. Quand celles-ci sont assemblées, *Internet des Objets* signifie sémantiquement « un réseau mondial d'objets interconnectés uniquement adressables, basé sur les protocoles standards de communication » [20]. Ainsi, la troisième vision, orientée « sémantique » (*"Semantic"-oriented*), concerne comment donner une adresse unique à chaque objet et comment représenter et stocker les données qui sont échangées sur le réseau IoT. Le paradigme de l'*Internet des Objets* est donc le résultat de la convergence entre ces trois visions. [20]

1.3.1.1 Vision orientée « objets »

La vision « objets » ne concernait au début qu'uniquement les tags *Radio-frequency Identification (RFID)*, mais les technologies *Near Field Communications (NFC)* et *Wireless Sensor and Actuators Networks (WSAN)* sont aussi devenus des éléments élémentaires de l'IoT. Des concepts plus abstraits ont également émergé,

1. C'est l'acronyme IoT qui est principalement utilisé à travers ce document.

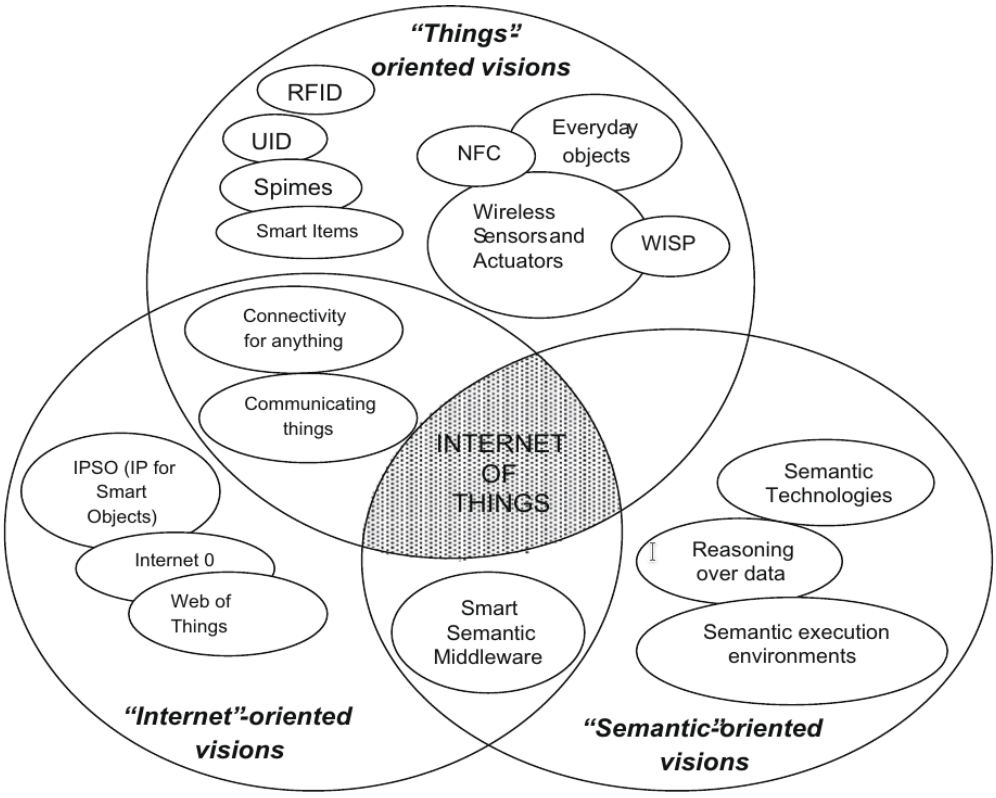

Figure 1.9 – Les 3 visions du paradigme de l'Internet des Objets.
Source: [20]

comme le *spime* qui se définit comme un objet qui peut être suivi à travers l'espace et le temps durant toute sa vie et qui sera durable, améliorable et identifiable de manière unique. Ce concept trouve une implémentation dans le monde physique avec les *Smart items*, vus comme des sortes de capteur équipé avec diverses technologies telle que la communication sans-fil, de la mémoire et d'autres capacités de calcul. [20]

En allant encore plus loin, l'ITU (Union internationale des télécommunications) voient l'IoT comme une étape au-delà de la connexion pour « n'importe qui, n'importe où », puisqu'il s'agirait maintenant d'avoir une connexion pour « n'importe quoi ». Le consortium CASAGRAS voit même l'IoT comme une infrastructure permettant de connecter les objets virtuels et ceux du monde physique. [20]

1.3.1.2 Vision orientée « réseau »

Cette vision cherche principalement a concrétisé les connexions nécessaires entre objets de l'IoT. Elle nous vient de l'alliance IPSO (IP for Smart Objects), qui considère le protocole IP, connectant déjà un très grand nombre d'appareils à travers le monde, comme le moyen le plus approprié pour déployer l'IoT. Ainsi, les initiatives comme 6LoWPAN (décrit comme une couche d'adaptation entre la couche réseau et la couche de liaison de la suite de protocoles TCP/IP) permettent de connecter un grand nombre d'éléments pouvant posséder très peu de ressources, que se soit au niveau du processeur, de la mémoire ou de la batterie [21].

1.3.1.3 Vision orientée « sémantique »

La troisième vision vient du constat que le nombre d'éléments impliqués dans l'IoT sera très élevé et les données générées par ceux-ci seront également en très grande quantité. Il y a donc un réel besoin en termes de représentation, de stockage, d'analyse, etc. des données produites. C'est par rapport à ce besoin que les technologies sémantiques doivent jouer un rôle important afin de donner du sens à des informations pouvant être disparates. [20]

1.3.2 Applications

L'IoT offre de très nombreuses possibilités d'applications. Pour citer quelques exemples parmi beaucoup d'autres, l'IoT pourrait être utile dans le domaine du transport et de la logistique, le domaine des soins de santé, le domaine des environnements « intelligents » ou le domaine personnel et social [20].

En ce qui concerne le domaine du transport et de la logistique, nous pouvons déjà trouver sur nos routes des voitures équipées d'une panoplie de capteurs et d'actionneurs, qui peuvent fournir au conducteur des informations utiles permettant une meilleure conduite et une sécurité accrue [20] (par exemple, grâce aux systèmes de détection de collision ou de franchissement de ligne). Un exemple souvent mentionné dans la littérature est celui du transport de marchandise comme les fruits, la viande ou les produits laitiers. Avec l'IoT et l'équipement nécessaire, il est possible de contrôler l'état de conservation (température, humidité, secousse, ...) de la marchandise et prendre les actions nécessaires avant d'éviter des déchets [22].

Dans le domaine des soins de santé, les technologies IoT pourraient permettent de suivre les patients et leurs habitudes au quotidien. En effet, des appareils portables peuvent collecter certains indicateurs en temps-réel reflétant la condition d'un patient.

Les smart homes ou smart cities sont des exemples typiques d'environnements « intelligents ». Dans le cas d'une smart home équipée de capteurs et actionneurs, il est possible d'améliorer la sécurité et le confort des habitants. Par exemple, la lumière ou le chauffage peuvent être adaptés en fonction du moment de la journée, les infractions peuvent être évitées en adoptant un système de monitoring et d'alarme.

Enfin, les applications concernées par le dernier domaine, personnel et social, ont pour but de permettre aux utilisateurs d'interagir avec d'autres personnes pour maintenir et construire des relations sociales. Par exemple, des objets connectés pourraient envoyer automatiquement à des amis des informations sur l'endroit où une personne se trouve, sur ce qu'elle fait, etc. [20].

1.3.3 « Context-awareness »

Le concept de « *Context-awareness* » est important pour la suite de ce mémoire, principalement dans le cadre de la solution proposée au chapitre 3. Le terme *contexte* peut être défini de la façon suivante :

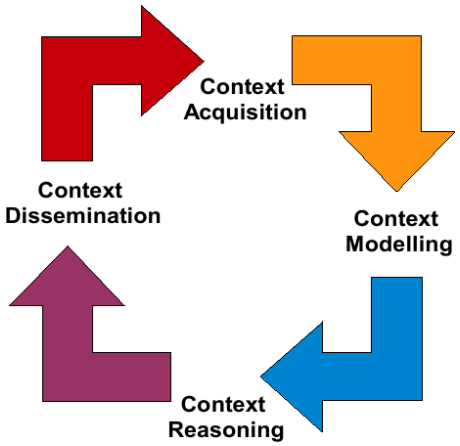

Figure 1.10 – Cycle de vie du contexte.
Source: [24]

> *« Le contexte est n'importe quelle information qui peut être utilisée pour caractériser la situation d'une entité. Une entité est une personne, un endroit, ou un objet qui est considérée pertinente à l'interaction entre un utilisateur et une application, y compris l'utilisateur et les applications elles-mêmes. »* [23]

Le terme *context-awareness*, traduit littéralement *conscience du contexte*, se définit comme la capacité de fournir des informations ou des services pertinents à l'utilisateur en se basant sur le contexte [24]. Il existe deux types de contextes :

— Contexte *primaire* : Information récupérée sans utiliser ce qui est déjà connu sur le contexte et sans faire aucune sorte de fusion des données provenant des capteurs. Par exemple, utiliser des données GPS en tant que données de localisation [24].

— Contexte *secondaire* : Information qui peut être calculée à partir du contexte primaire. Elle peut être calculée en fusionnant des données provenant des capteurs ou en utilisant des données récupérées d'un service web. Par exemple, identifier la distance entre deux points en utilisant deux données GPS du contexte primaire [24].

De plus, dans [24], un cycle de vie du contexte, composé de quatre étapes, est défini. La premières étape consiste à acquérir le contexte à partir de différentes sources, comme des capteurs. Ensuite, la donnée doit être modélisée/représentée de manière pertinente. La troisième étape a pour but de traiter les données pour dériver une information contextuelle de haut-niveau à partir des données brutes (de bas-niveau) provenant des capteurs. Pour terminer, la quatrième étape propage les informations contextuelles (de haut- et bas-niveau) aux utilisateurs qui souhaitent en faire utilisation. Une illustration de ce cycle de vie se trouve à la figure 1.10.

1.3.4 Lien avec le proactive computing

Une des trois caractéristiques citées par David Tennenhouse dans son article sur le proactive computing, *Getting physical*, fait trivialement penser au paradigme de l'IoT. En effet, cette caractéristique nous dit que les systèmes proactifs doivent être connectés au monde physique qui les entoure et qu'ils devront être capables d'utiliser les capteurs et les actionneurs pour, à la fois, surveiller et modéliser leur environnement [12].

Les systèmes IoT semblent plus que convenir pour remplir ce rôle nécessaire aux systèmes proactifs, puisqu'ils peuvent fournir un réseau de capteurs et d'actionneurs très variés permettant de récupérer diverses informations sur le contexte environnant. Ainsi, un système proactif pourrait exploiter de manière efficace toutes ces informations, notamment en étant capable d'en retirer des données appartenant au contexte secondaire tel qu'expliqué au chapitre 3.

Chapitre 2

Problématique

La force principale du modèle ABAC est sa flexibilité, qui réside principalement dans le fait de pouvoir utiliser un ensemble de multiples et divers attributs dans les politiques. S'il était possible d'alimenter cet ensemble avec des données venant d'un système IoT, le modèle ABAC pourrait montrer tout son potentiel. En effet, il serait dès lors possible d'utiliser toute la diversité que peut fournir un tel système et ainsi renforcer et peaufiner les politiques de sécurité.

Par exemple, prenons une pièce équipée de divers capteurs s'occupant de monitorer la température, l'humidité, la détection de présence, etc. En stockant les données que ces capteurs fournissent, dans une base de données par exemple, et en les rendant accessibles via la PIP, il serait possible de concevoir des politiques de sécurité en utilisant XACML qui régiront l'accès à la pièce en s'appuyant sur des informations contextuelles. De plus, comme il n'existe pas de restriction ou d'imposition en ce qui concerne le remplissage de la source de données derrière le PIP, il est tout à fait envisageable de traiter les données brutes des senseurs, à l'image de la définition du contexte secondaire de la section 1.3.3, et de remplir cette source avec le résultat du traitement.

2.1 Limites du modèle ABAC

Ainsi, combiner ABAC et IoT pourrait offrir bien des avantages, mais pour concrétiser cette combinaison, deux problèmes, illustrés à la figure 2.1, se posent :

— Comment alimenter le PIP avec des données venant d'un système IoT ?

— De quelle manière agir sur le monde physique lorsque c'est nécessaire ?

2.1.1 Alimenter les sources d'attributs

Alimenter le PIP peut simplement se faire en insérant de l'information dans une base de données. Cependant, la difficulté vient du fait qu'il est nécessaire de récupérer préalablement ces données et de les stocker de manière « intelligente » pour ne pas se cantonner à insérer les données brutes, alors qu'il est possible d'en tirer des métadonnées en les combinant/agrégeant, mais aussi d'éviter de stocker une donnée peut-être non pertinente.

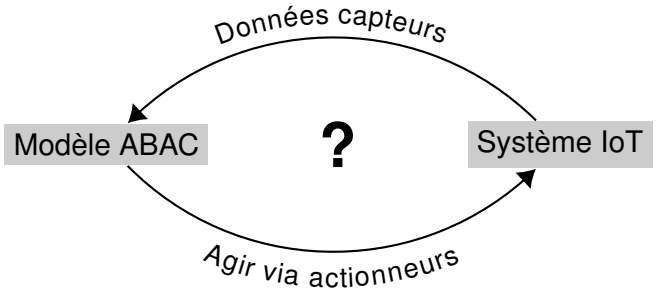

Figure 2.1 – Illustration de la problématique.

Notons qu'il ne s'agit pas vraiment d'une limite du modèle ABAC qui est soulignée ici, puisque le modèle prend en compte la diversité dont les sources d'attributs du PIP peuvent faire preuve, en plaçant le PIP comme une interface permettant de ne pas se soucier des détails d'implémentation de ces différentes sources. Mais si nous voulons avoir la possibilité de baser fortement les politiques de sécurité sur les attributs reflétant l'état de l'environnement, un système est nécessaire pour faire le lien entre le modèle ABAC et un système IoT. Cependant, un tel système peut s'avérer laborieux à implémenter et peut-être difficile à généraliser pour ne pas fonctionner qu'uniquement avec un système particulier.

2.1.2 Agir sur l'environnement

Si utiliser le modèle ABAC avec un système IoT semble être une combinaison très intéressante, pouvoir agir sur l'environnement via les actionneurs constitue une fonctionnalité supplémentaire non négligeable. Néanmoins, il n'est pas possible d'interagir de manière synchrone sur l'environnement en utilisant uniquement ABAC.

Remarquons qu'il existe tout de même le mécanisme d'obligation, décrit à la section 1.1.4.1. Pour rappel, celui-ci permet de faire exécuter certaines opérations par le PEP quand une réponse, concernant l'autorisation d'un accès, est renvoyée par le PDP. Cela pourrait permettre d'agir sur l'environnement si le PEP en a la capacité, mais seulement quand une réponse à une demande d'accès est retournée par PDP, et non pas lorsque quelque chose change au sein de l'environnement.

2.2 Quid du proactive computing ?

En théorie, nous remarquons donc que combiner ABAC et IoT peut s'avérer une combinaison puissante, mais encore faut-il pouvoir trouver une manière idéale de la réaliser et surmonter les deux « obstacles » décrits ci-avant. Pour ce faire, il a été choisi de tenter l'utilisation du proactive computing. Ainsi, la question centrale de ce mémoire est la suivante :

> *Comment utiliser le proactive computing pour concrétiser une combinaison entre le modèle ABAC et un système IoT ?*

De celle-ci, deux sous-questions sont implicitement dérivées :

— *Est-il possible d'utiliser le proactive computing pour alimenter le modèle ABAC en données provenant d'un système IoT ?* De préférence, ces données devraient être pré-traitées afin de pourvoir utiliser des politiques d'un plus haut-niveau d'abstraction.

— *Le proactive computing peut-il nous permettre d'agir de manière efficace sur l'environnement lorsque cela est nécessaire ?* Le fait d'agir sur l'environnement devrait notamment se faire en réponse à des événements au sein de ce dernier, et idéalement sans que l'intervention d'une personne soit nécessaire.

Chapitre 3

Application du proactive computing

Dans ce chapitre, une solution possible à la problématique exposée au début de ce chapitre, qui n'a pas été retenue comme solution finale, est abordée dans un premier temps à la section 3.1. Ensuite, la version retenue est décrite en détail à la section 3.3, suivie par la section 3.4 insistant sur un aspect fondamental de la solution, la communication entre composants, et par la section 3.5 détaillant le lien entre le moteur proactif et le modèle ABAC.

3.1 Exploration d'une solution possible

Une solution possible qui n'a pas été retenue est d'utiliser le moteur proactif en tant que gestionnaire des politiques d'accès dans l'architecture XACML. Le moteur proactif agirait alors en utilisant le PAP. Pour rappel, le rôle de ce dernier est de fournir une interface pour permettre à des administrateurs de gérer (ajouter, supprimer, modifier) les politiques. Le moteur proactif serait en quelque sorte identifié à un administrateur, qui s'occuperait de garder à jour les politiques d'accès. Il les adapterait principalement en fonction des informations remontées par les capteurs du système IoT.

Ainsi, l'idée est qu'il ne serait plus nécessaire pour des administrateurs d'écrire et maintenir les politiques. L'humain pourrait donc ne plus être impliqué dans la gestion du système de sécurité. Bien que cela semble intéressant d'un point de vue de l'« automatisation », cela implique que le moteur proactif devra faire preuve de plusieurs caractéristiques, notamment la génération et modification des politiques. Cependant, même si cela semble intéressant, il paraît laborieux d'implémenter un tel système en utilisant le proactive computing. En effet, il faudrait vérifier que les politiques sont toujours complètes et suffisantes, puisque nous voulons éviter l'intervention d'un administrateur.

De plus, retirer complètement l'administrateur de la gestion peut être quelque peu exagéré, puisque celui-ci perdrait le contrôle et la flexibilité offerts par le mo-

Figure 3.1 – Moteur proactif comme middleware IoT.

dèle ABAC. Même si la tâche de l'administrateur se verrait allégée, il sera tout de même nécessaire d'implémenter des règles pour le moteur proactif, ce qui s'apparente à simplement transférer le problème. Il semblerait plus judicieux de laisser un administrateur superviser le système de sécurité via la gestion des politiques. En outre, les politiques ne sont généralement pas amenées à être modifiées très souvent, du moins pas aussi rapidement que l'état de l'environnement. Par contre, les attributs reflétant l'état de l'environnement sont implicitement propices à des modifications fréquentes. Le moteur proactif devrait donc agir d'une autre manière afin de combiner le modèle ABAC et un système IoT.

3.2 Notion de middleware IoT

Une notion que nous retrouvons dans la littérature, importante pour la suite de ce chapitre et pour la façon dont le moteur proactif devrait être utilisé pour répondre à la problématique posée au chapitre 2, est celle de *middleware*. Cette notion peut se définir de la façon suivante :

> *« Le middleware est une couche logicielle ou un ensemble de sous-couches interposées entre le niveau technologique et le niveau applicatif. Sa caractéristique de cacher les détails des différentes technologies est fondamentale pour exempter le programmeur de questions qui ne sont pas directement pertinentes à son objectif, qui est le développement de l'application spécifique permise par les infrastructures de l'IoT. »* [20]

Dans la section suivante (3.3), la solution proposée place le moteur proactif en tant que middleware, permettant ainsi une liaison entre le modèle ABAC (niveau applicatif) et un système IoT (niveau technologique), comme l'illustre la figure 3.1. Il sera utilisé de manière à fournir un moyen facilitant l'interaction avec un réseau de senseurs et d'actionneurs. Par conséquent, les règles du moteur proactif permettront de faciliter la programmation d'un système exploitant les données de ce réseau. En effet, de par sa nature basée sur des règles, le moteur proactif offre un framework facilitant la collecte des données provenant du système IoT, ainsi que les calculs pouvant être faits sur celles-ci.

3.3 Architecture logique proposée

La solution élaborée autour du moteur proactif s'architecture tel que le montre la figure 3.2. Celle-ci se divise en trois parties principales : la partie qui concerne

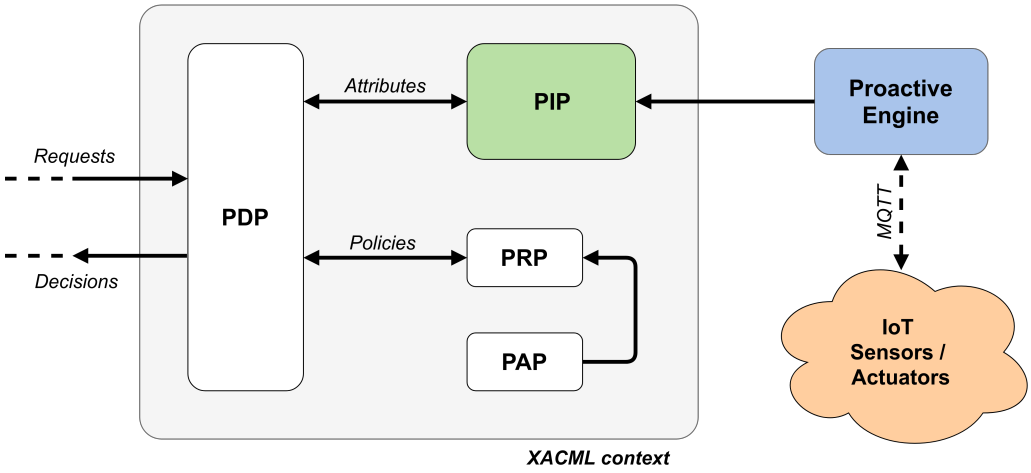

FIGURE 3.2 – Architecture logicielle de la solution.

le contrôle d'accès et en particulier le modèle ABAC, le moteur proactif et le système IoT. Mis à part quelques caractéristiques supplémentaires, la partie relative au contrôle d'accès ne varie pas par rapport à celle définie par le modèle ABAC (expliquée à la section 1.1.4.1), les mêmes composants s'y retrouvent donc. En ce qui concerne le système IoT, les capteurs et actionneurs qui le composent peuvent être très variés, ils ne sont donc pas détaillés ici. L'unique contrainte faite sur les composants du système IoT est la façon dont ils doivent communiquer avec le moteur proactif, ceci est expliqué à la section 3.4. Enfin, le cœur de la solution proposée est le moteur proactif. C'est lui qui permet de répondre à la problématique posée au chapitre 2 et plus précisément aux deux sous-problèmes que sont : comment fournir la source de données derrière le PIP avec des données venant du système IoT et comment agir sur l'environnement lorsque c'est nécessaire.

Notons que par rapport à la solution proposée à la section 3.1, celle proposée dans cette section se veut moins « invasive » puisqu'il est toujours possible laisser une personne gérer les politiques via le PAP, comme prévu initialement dans le modèle ABAC. L'architecture ABAC ne doit donc pas être modifiée fondamentalement. En outre, l'avantage important est qu'il sera possible d'utiliser, dans ces politiques, des attributs dont la valeur reflète l'état d'un certain aspect de l'environnement. Les politiques pourront dès lors être plus précises et plus complètes, en utilisant le plus d'informations possible.

3.3.1 Rôles des composants principaux

3.3.1.1 PDP

Le PDP est toujours chargé de prendre les décisions par rapport aux demandes d'accès. Ces demandes émaneront d'un des PEP, qui ne sont intentionnellement pas représentés sur la figure 3.2, car ceux-ci peuvent se retrouver à plusieurs endroits à travers un système. Cette variabilité vient du fait que c'est le PEP qui doit effectivement appliquer les politiques de sécurité à toutes les ressources qui requièrent une forme de protection. Par exemple, il pourrait faire partie d'un gateway d'accès à distance ou d'un serveur Web [10]. Dans le contexte IoT, il pourrait même être

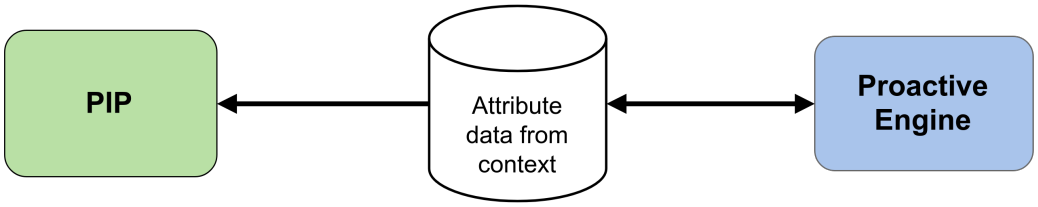

Figure 3.3 – Lien concret entre PIP et moteur proactif.

couplé à un système de serrure pour porte utilisant la technologie RFID, où les accès à une pièce seraient régis par des politiques XACML.

Notons que dans ce cas-ci, l'aspect supplémentaire apporté au PDP est qu'il évaluera des politiques pouvant fortement référencer des attributs relatifs à l'environnement. Il s'appuiera donc sur le PIP pour récupérer les valeurs au moment de l'évaluation. En effet, lorsqu'une politique référence un attribut dont la valeur n'est pas disponible dans le contexte direct de l'évaluation, un mécanisme au sein du PDP est chargé de récupérer la valeur courante de cet attribut, en exécutant les requêtes nécessaires auprès du PDP.

3.3.1.2 PIP

Entre tous les composants du modèle ABAC, celui-ci est le plus important dans le cadre de ce mémoire. En effet, du fait que c'est lui qui a la tâche de stocker les valeurs des attributs et de les rendre disponible au PDP, il sera utilisé comme passerelle vers l'architecture ABAC par le moteur proactif. Le moteur proactif aura donc une influence sur cette dernière à travers le PIP.

Plus concrètement, le moteur proactif aura accès à une base de données commune entre ce dernier et le PIP. Cette base de données est donc le lien concret entre ces deux composants principaux, comme l'illustre la figure 3.3.

La façon dont se structure la base de données à laquelle le PIP et le moteur proactif se connectent dépend bien entendu du contexte applicatif. De plus, comme le spécifie le standard XACML, le PIP permettra d'abstraire cette base de données en transformant les données qu'elle contient de manière a ce que le PDP puisse en faire l'usage lors d'une évaluation.

3.3.1.3 PAP et PRP

En ce qui concerne la partie liée au contrôle d'accès, ces deux derniers composants remplissent encore une fois le même rôle que dans l'architecture initiale. Pour rappel, le PAP permet de gérer l'ensemble des politiques stockées dans le PRP. Ce travail n'a pas pour but d'altérer le comportement de ces deux composants, c'est pourquoi leur fonctionnement ne sera pas détaillé. Par contre, la façon d'écrire des politiques XACML et plus particulièrement comment référencer un attribut externe (par exemple, une caractéristique de l'environnement reflétée par le système IoT), constitue des caractéristiques qu'il est important de considérer et qui seront donc abordés à la section 3.5.

3.3.1.4 Moteur proactif

Comme expliqué au début de la section 3.3.1.2, le moteur proactif aura accès à une base de données commune entre le PIP et lui-même. Les règles du moteur proactif pourront ainsi exécuter toutes les actions nécessaires (lire, écrire, modifier, supprimer) pour garder celle-ci à jour par rapport à la situation courante de l'environnement, reflétée à travers les capteurs du système IoT dont il est question. Par exemple, il est possible d'imaginer un scénario où la première règle pourrait être chargée de récupérer les données auprès des capteurs. Ensuite, une deuxième règle (appartenant toujours au même scénario) pourrait réaliser certains traitements sur ces données, comme les combiner pour en retirer une métadonnée. Le résultat de ce traitement serait alors sauvegardé, grâce à une troisième règle, dans la base de données utilisée par le PIP.

Le rôle du moteur proactif dans cette architecture logicielle fait clairement référence à la notion de middleware telle que décrite à la section 3.2. Cependant, afin de remplir ce rôle pleinement, le moteur proactif doit aussi disposer d'un moyen de communication, aussi bien pour récupérer des données, que pour agir sur l'environnement rendu accessible à travers les actionneurs du système IoT. Ce moyen de communication est décrit en détail à la section 3.4.

3.3.1.5 Système IoT

Dans l'architecture proposée, il est difficile de définir exactement le système IoT puisque ce dernier peut être composé de capteurs et d'actionneurs multiples et différents. Il semble cependant utile de bien définir ce dont il est question lorsque les termes capteur et actionneur sont utilisés dans ce travail.

Un capteur est considéré comme un composant matériel qui capture des informations sur le monde physique en répondant à un stimulus physique [25]. Un tel dispositif peut donc être utilisé pour détecter et mesurer de la lumière, de la chaleur, de l'humidité ou toute autre caractéristique physiquement mesurable. Une taxonomie des capteurs qu'il est possible de retrouver dans un système IoT est disponible à la table 3.1. Les différentes colonnes de cette table se définissent de la façon suivante [26] :

— Motion : Englobe les mesures liées au mouvement d'un corps qui peut être n'importe quel type de masse (solide, liquide ou gazeuse, animée ou inanimée).

— Position : Concerne les mesures relatives à la position d'une entité physique.

— Environnement : Tout ce qui peut être mesuré par rapport à un environnement physique.

— Mesure : Tout ce qui peut être mesuré à partir d'une entité physique ou à partie d'une force d'interaction avec une entité.

— Bio-capteur : Regroupe les capteurs permettant de prendre des mesures sur un organisme biologique.

Un actionneur se défini comme un composant matériel qui manipule le monde physique. Les actionneurs reçoivent typiquement des commandes qu'ils traduisent

Type	**Motion**	**Position**	**Environnement**	**Mesure**	**Bio-capteur**
Sous-type	Mouvement	Orientation	Température	Volume	Sang
	Vélocité	Inclinaison	Humidité	Pression	Organe
	Inertie	Proximité	Luminance	Densité	Mental
	Vibration	Présence	Acoustique	Déformation	Tissu
	Accélération	Localisation	Rayonnement	Viscosité	
	Rotation		Gaz	Flux	
			Champ magnétique	Charge	
			Météo	Humidité	
			Chimique	Choc	
			Électrique	Contact	
			Couleur	Souche	
			Champ électro-magnétique	Corrosion	
				Conductivité électrique	
				Oxygène	

Table 3.1 – Types de capteurs dans l'IoT.
Source: [26]

ensuite en signaux électriques pour réaliser une forme d'action physique. Par exemple, un actionneur pourrait allumer ou éteindre le système de ventilation d'une pièce pour y faire varier l'humidité. [25]

Enfin, un capteur ou un actionneur ne possède généralement pas directement la capacité d'exécution de logiciel. Cette capacité est rendue possible en utilisant un dispositif complémentaire. Un tel dispositif est un composant logiciel qui est connecté (avec ou sans fil) à un ou plusieurs capteurs/actionneurs ou qui intègre directement un ou plusieurs capteurs/actionneurs. Il constitue donc le point d'entrée du monde physique vers le monde digital, et c'est donc aussi lui qui offre des possibilités de communication (en passant parfois par un gateway), constituant une caractéristique très importante pour permettre au moteur proactif de communiquer avec un système IoT tel qu'expliqué à la section suivante (3.4). [25]

3.4 Communication entre moteur proactif et système IoT

La communication entre le moteur proactif et un système IoT constitue l'un des aspects les plus importants de la solution décrite ci-avant, car il est essentiel au bon fonctionnement de l'architecture logique proposée. Dans le monde de l'IoT plusieurs

technologies et protocoles peuvent être utilisés, il a donc fallu faire certains choix, expliqués dans le détail de cette section, pour réaliser quelque chose de concret avec le moteur proactif. Il va de soi que la façon dont la communication entre le système IoT et le moteur proactif est établie dépend plus de la nature du système IoT que du moteur en lui-même puisque ce dernier est assez facilement extensible et modifiable. En revanche, il n'existe pas vraiment de façon universelle pour faire communiquer les capteurs et actionneurs d'un système IoT avec le monde digital, dû à l'hétérogénéité des types de matériel, c'est pourquoi il a fallu trancher entre les différentes possibilités que se présentaient, tout en pesant le pour et le contre de chacune.

Remarquons que cette section répond à la deuxième question de la problématique posée au chapitre 2, à savoir comment agir sur l'environnement via les actionneurs ; et en partie à la première question qui concernait la possibilité d'utiliser les données provenant d'un système IoT dans les politiques de sécurité.

3.4.1 Protocoles dans l'IoT

Si nous nous intéressons aux différentes couches réseaux qui composent l'écosystème IoT, nous remarquons très vite toute la diversité qui se présente lors de la conception d'un système IoT. En effet, au niveau de la couche de *liaison de données* et *physique*, il est possible de citer une multitude de protocoles/technologies utilisés à ce niveau, dont les plus fréquents dans le monde de l'IoT :

IEEE 802.15.4 Il s'agit là d'un des protocoles le plus adaptés dans le monde de l'IoT, puisqu'il permet la communication entre plusieurs appareils qui se veulent compacts, qui possèdent une source d'alimentation faible et qui doivent faire preuve d'une longue durée de vie de la batterie. [27]

IEEE 802.11ah Ce protocole découle directement du protocole IEEE 802.11, plus connu sous le nom de Wi-Fi. Il a été conçu dans le but de permettre l'utilisation du Wi-Fi dans l'IoT. Comme la norme IEEE 802.15.4, il permet la communication entre appareils à faibles ressources, notamment en opérant à une fréquence inférieure à 1 GHz. [28]

Bluetooth Low Energy Pour faire face au besoin grandissant de protocole M2M (Machine à Machine), la technologie Bluetooth a elle aussi eu droit à une déclinaison moins énergivore. [29]

Plus concrètement, il est possible de lister plusieurs technologies, dont l'utilisation dépendra notamment de la distance qui sépare les appareils voulant être connectés. Afin d'être complet, la figure 3.2 reprend un bon nombre d'entre elles. Il n'est pas réellement utile de les expliquer en détail dans ce document, puisque ce n'est pas à ce niveau qu'un choix doit être réalisé. En effet, la technologie utilisée dépendra principalement du constructeur des capteurs et actionneurs, et la solution développée dans le cadre de ce mémoire ne cherche pas à poser de contrainte à ce niveau.

Ainsi, à la couche *réseau*, le nombre de protocoles est plus succinct. En effet, l'IPv6 peut déjà être utilisé comme moyen d'adressage et de paquetage pour les appareils de l'IoT. De plus, dans le but d'optimiser l'IPv6 dans le cadre d'une

Technologie	Fréquence	Débit	Portée	Puissance	Coût
2G/3G	Bandes cellulaires	10 MB/s	Plusieurs km	Haute	Haut
802.15.4	2.4 GHz	250 kb/s	100 m	Faible	Faible
Bluetooth	2.4 GHz	1, 2.1, 3 Mb/s	100 m	Faible	Faible
LoRa	< 1 GHz	< 50 kb/s	2–5 km	Faible	Moyen
LTE Cat 0/1	Bandes cellulaires	1–10 Mb/s	Plusieurs km	Moyenne	Haut
NB-IoT	Bandes cellulaires	0.1–1 Mb/s	Plusieurs km	Moyenne	Haut
SIGFOX	< 1 GHz	Très faible	Plusieurs km	Faible	Moyen
Weightless	< 1 GHz	0.1–24 Mb/s	Plusieurs km	Faible	Faible
Wi-Fi (11f/h)	2.4, 5, < 1 GHz	0.1–1 Mb/s	Plusieurs km	Moyenne	Faible
WirelessHART	2.4 GHz	250 kb/s	100 m	Moyenne	Moyen
ZigBee	2.4 GHz	250 kb/s	100 m	Faible	Moyen
Z-WAVE	908.42 MHz	40 kb/s	30 m	Faible	Moyen

Table 3.2 – Technologies sans fil utilisées dans l'IoT.
Source:
`http://www.electronicdesign.com/iot/12-wireless-options-iotm2m-diversity-or-dilemma`

utilisation axée sur l'IoT, la standard 6LoWPAN a été introduit par l'Internet Engineering Task Force (IETF). Il permet principalement l'interopérabilité entre la couche de *liaison de données* (où le protocole IEEE 802.15.4 opère) et le protocole IPv6, en fournissant trois fonctionnalités principales : la compression des en-têtes, la segmentation et le réassemblage des paquets, et le transfert vers la couche inférieure (de *liaison de données*). [21]

L'alternative la plus populaire à la pile IPv6, 6LoWPAN et IEEE 802.15.4 est le protocole ZigBee. Il se base également sur le protocole IEEE 802.15.4, mais il couvre également les couches supérieures (de la couche *réseau* jusqu'à une partie de la couche *applicative*). [30] L'interopérabilité avec d'autres protocoles est donc rendue plus difficile [31], c'est principalement pourquoi, dans notre cas, il n'a pas été choisi comme moyen de communication avec le système IoT avec lequel le moteur proactif devrait interagir.

Dû au fait que les technologies et protocoles des couches bas niveaux dépendent fortement du type d'appareils utilisés et des constructeurs qui les fournissent, le choix doit donc être fait à un autre niveau, c.-à-d. au niveau de la couche *applicative*. Pour faire ce choix, il est d'abord nécessaire de comparer deux paradigmes de communication, ce qui est fait à la section 3.4.1.1 ci-après.

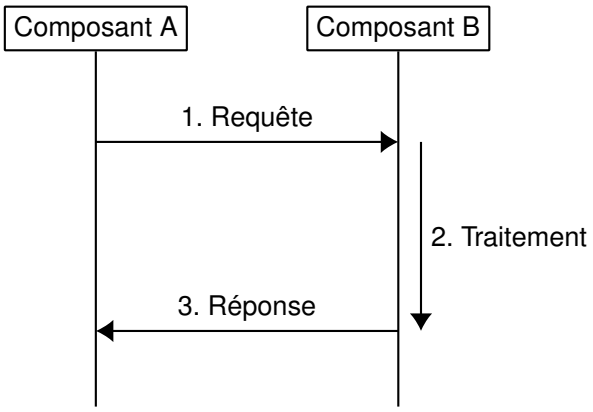

FIGURE 3.4 – Paradigme de communication Requête – Réponse.
Source: [32]

3.4.1.1 Paradigmes de communication

Les protocoles de communication de la couche *applicative* dans le monde de l'IoT peuvent se classer en deux paradigmes.

Requête – Réponse

Ce paradigme permet la communication bidirectionnelle entre deux composants d'un réseau. Un composant initie la communication en envoyant un message à un autre composant cible, qui le réceptionne et l'exploite. Ce dernier répond en envoyant un nouveau message dans l'autre sens (vers le composant qui a initié la communication). Ce mode de fonctionnement est illustré à la figure 3.4. [32]

Ce paradigme se prête bien à l'IoT dans les cas suivants [32] :

1. L'architecture cible suit un pattern *client – serveur*.
2. Une communication interactive est requise, où les composants doivent s'envoyer des informations mutuellement.
3. La réception d'informations doit être confirmée par un accusé de réception.

Toutes les applications n'ont pas nécessairement besoin de toutes ces caractéristiques, qui peuvent constituer une surcharge non nécessaire. Le paradigme Publish – Subscribe fournit dès lors une alternative.

Publish – Subscribe

Contrairement au paradigme Requête – Réponse, le paradigme Publish – Subscribe propose des communications unidirectionnelles, comme l'illustre la figure 3.5. Un publisher publie un message correspondant à une classe ou une catégorie particulière. Un subscriber peut exprimer son intérêt par rapport à une classe/catégorie, et ainsi recevoir les messages qui y correspondent, lorsqu'un publisher (postant des messages de cette classe/catégorie) possède une nouvelle donnée. [32]

Ainsi, ce paradigme offre les différentes caractéristiques (en plus de l'unidirectionnalité) :

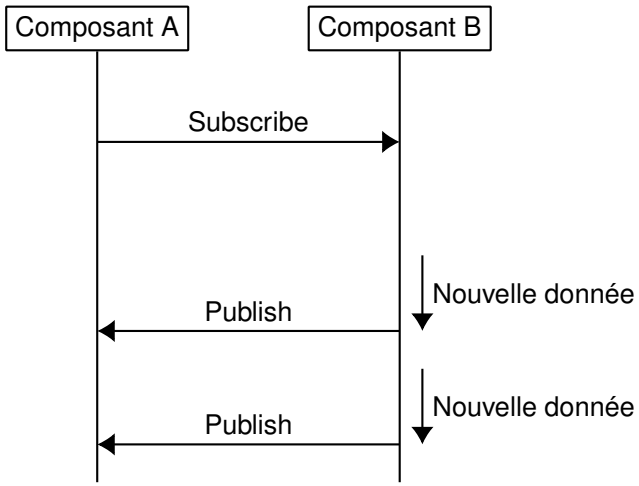

Figure 3.5 – Paradigme de communication Publish – Subscribe.
Source: [32]

Protocoles	Protocole couche *transport*	Paradigme
XMPP	TCP	Les deux
AMQP	TCP	Publish/Subscribe
CoAP	UDP	Requête/Réponse
MQTT	TCP	Publish/Subscribe

Table 3.3 – Protocoles de la couche *applicative* dans l'IoT.

1. Un faible couplage entre les composants du réseau.
2. Une meilleure capacité de mise à l'échelle en tirant profit du parallélisme et des capacités de multicast.

3.4.1.2 Protocoles M2M

À l'instar des protocoles de la couche *liaison de données* et *physique*, il est possible de citer un bon nombre de protocoles (machine à machine) opérant à la couche *applicative*. La table 3.3 liste les protocoles les plus répandus et reprend, d'une part le protocole de la couche *transport* sur lequel s'appuie le protocole de la couche *applicative* dont il est question, et d'autre part le paradigme auquel il correspond.

En considérant les deux paradigmes définis à la section précédente, dans le cadre du moteur proactif, utiliser le paradigme Requête – Réponse ne semble pas être le choix le plus judicieux. En effet, il est plus question ici de communication unidirectionnelle que bidirectionnelle, puisque le moteur collectera les données des capteurs et enverra des commandes aux actionneurs. C'est donc bidirectionnel lorsque le système IoT est pris dans son ensemble, mais toujours unidirectionnel lors de la communication avec soit un capteur (du capteur au moteur proactif), soit un actionneur (du moteur proactif vers l'actionneur). De plus, s'il est souhaité que différentes règles puissent recevoir les informations d'un même capteur, utiliser le

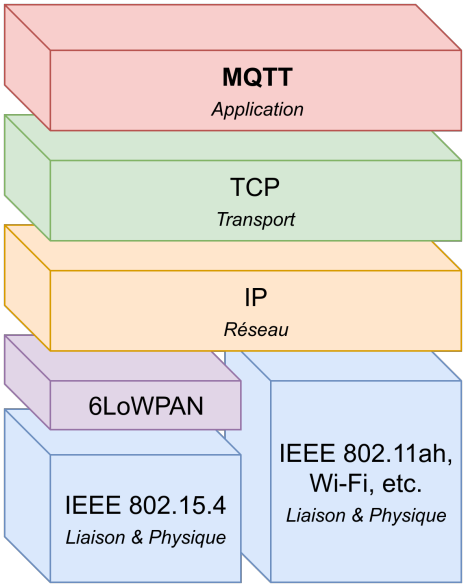

FIGURE 3.6 – Pile protocolaire dans le cadre de la solution proposée.

paradigme Requête – Réponse signifierait qu'une connexion avec chaque règle devra être initiée, ce qui peut engendrer une surcharge importante. Pour ces raisons, le protocole CoAP ne sera pas utilisé dans le cadre du moteur proactif.

Le choix doit donc se tourner vers AMQP, XMPP ou MQTT. À ce stade, il faut trancher entre les trois et MQTT a été choisi pour une raison principale : son adoption assez répandue et la richesse des librairies disponibles implémentant ce protocole. Enfin, pour résumer les sections précédentes, la pile de protocoles dans le cas de la solution proposée pour répondre à la problématique peut se résumer par la figure 3.6. Au niveau des protocoles de la couche *transport* et de la couche *réseau*, respectivement le protocole TCP et le protocole IP, s'imposent dû au choix du protocole MQTT pour la pile applicative, puisque ce dernier se base sur TCP/IP.

3.4.2 MQTT

MQTT est un acronyme pour « Message Queue Telemetry Transport ». Ce protocole « machine à machine » a été créé par IBM et a été standardisé par la suite par OASIS. La dernière version, la 3.1, date de décembre 2015. C'est un protocole asynchrone basé sur le pattern Publish – Subscribe qui fonctionne par-dessus le protocole TCP.

L'aspect central du protocole est le *broker* par lequel transitent tous les messages. Les *publishers* envoient des messages au *broker* sur un certain *topic*. Les *subscriber* peuvent exprimer leur intérêt sur un topic auprès du *broker*. Ainsi, lorsque le *broker* reçoit un message, il dispatche aux règles souscrites au *topic* sur lequel le message est arrivé. Le protocole est pensé pour utiliser le moins de bande passante possible et la batterie avec parcimonie. [33]

Le protocole fournit 3 niveaux de qualité de service (QoS – Quality of Service) [33] permettant de donner une priorité différente à certains messages :

1. Feu et oubli : Le message est envoyé une seule fois et aucun accusé de réception n'est requis.
2. Délivré au moins une fois : Le message est envoyé une seule fois et un accusé de réception est requis.
3. Délivré exactement une fois : Un « four-way handshake » est utilisé pour s'assurer que le message est délivré exactement une fois.

En ce qui concerne les *topics*, ils sont définis de manière hiérarchique et il existe deux types de « wildcards » permettant de définir un *filtre de topic*, qui rend possible la souscription à plusieurs *topics* [34] :

— Wildcard à plusieurs niveaux `"...//..."` : Permet de matcher plusieurs niveaux à l'intérieur d'un topic. Par exemple, `"sensors/#"` matchera avec `"sensors/temperature"`, `"sensors/humidity"`, `"sensors/temperature/id-01"`, etc.

— Wildcard à un seul niveau `"...//..."` : Permet de matcher un et un seul niveau à l'intérieur d'un topic. Par exemple, `"sensors/+/id01"` matchera avec `"sensors/temperature/id01"`, `"sensors/humidity/id01"`, etc.

D'un point de vue règle du moteur proactif, l'idée est qu'une règle pourrait souscrire, auprès du *broker* MQTT, à un ou plusieurs *topics* (via un *filtre de topic*) et ainsi recevoir des données venant des capteurs. Dans l'autre sens, une règle pourrait aussi jouer le rôle de *publisher*, en envoyant une commande à un ou plusieurs actionneurs.

3.4.3 MQTT et moteur proactif

Dans un souci de ne pas altérer en profondeur la version du moteur proactif existante, il a été choisi d'implémenter le broker de manière à ce qu'il n'interagisse pas directement avec le fonctionnement interne du moteur proactif, et donc qu'il n'interagisse pas avec les files internes au moteur. Le broker s'exécute en parallèle au moteur, sur un second thread. Concrètement, c'est la librairie *Moquette*[1] qui a été utilisée pour l'implémentation. Il s'agit d'une implémentation légère d'un broker MQTT réalisée en Java, ce qui facilite l'implémentation avec le moteur proactif existant puisque ce dernier est aussi développé avec le langage.

Ainsi, si nous nous intéressons au code Java nécessaire pour faire tourner un broker MQTT au côté du moteur proactif, un première classe abstraite a été définie telle qu'au listing 3.1. Les buts des différentes méthodes qui y sont définies sont les suivants :

— `start` : Cette méthode permet de démarrer le broker en l'initialisant avec les paramètres de configuration nécessaires.

— `stop` : Elle permet d'arrêter le broker MQTT et donc de fermer les différentes connexions.

1. `http://andsel.github.io/moquette/`

— `subscribeRule` : Souscris une règle à un certain topic. Le type de règle dont il est question ici est une extension du type de base défini à la section 1.2.2.

— `unsubscribeRule` : Antonyme de la méthode `subscribeRule`, elle permet de supprimer la souscription d'une règle à un certain filtre de topic.

— `sendMessage` : Donne la possibilité d'envoyer, directement via le broker, un message sur un certain topic.

Les deux autres méthodes (`flushMessages` et `setBufferActivated`) sont utilisées lors d'un mécanisme nécessaire par rapport au fonctionnement du moteur. Pour éviter des accès concurrents lors de l'exécution d'une règle, pendant une itération du moteur proactif, ou lors de la sauvegarde d'une règle en base de données, un buffer au sein du broker est utilisé pour accumuler les messages arrivés lors de l'itération courante. Ainsi, lorsqu'une règle a besoin d'accéder aux messages MQTT qu'elle a reçus, la liste contenant ces messages n'est pas modifiée. À la fin de l'itération, les règles sont sauvegardées en base de données avec leurs messages, puis les messages accumulés dans le buffer sont dispatchés entre les différentes règles connues, en fonction des topics auxquelles ces règles sont souscrites.

Listing 3.1 – Classe abstraite représentant un broker MQTT.

```
1  public abstract class MqttBroker {
2
3      public abstract void start();
4
5      public abstract void stop();
6
7      public abstract void subscribeRule(AbstractMqttRule rule);
8
9      public abstract void unsubscribeRule(AbstractMqttRule rule);
10
11     public abstract void sendMessage(String clientID, String topic,
       ↪  String message);
12
13     public abstract void flushMessages();
14
15     public abstract void setBufferActivated(boolean bufferActivated);
16
17 }
```

3.4.3.1 Un nouveau type de règle

Afin de faciliter la définition de scénarios utilisant des règles supportant la réception de messages MQTT (et l'envoi), un nouveau type de règle a été défini. Celui-ci est représenté par la classe abstraite définie au listing 3.2. Cette classe définit deux variables. La première, `messages`, a pour but de contenir les futurs messages MQTT que pourrait recevoir la règle. La seconde, `topicFilter`, contiendra le filtre de topics (sous la forme telle qu'expliquée à la section 3.4.2) permettant

de souscrire la règle au broker. Les getters et setters usuels sont aussi définis pour ces deux variables.

La méthode `execute` est redéfinie dans cette sous-classe de la classe `AbstractRule`, pour changer quelque peu son fonctionnement. En effet, le but d'une règle recevant des messages MQTT, qualifiée de règle de type MQTT, est de rester toujours à l'écoute de nouveaux messages, afin de réagir le plus vite possible aux changements au sein de l'environnement. Ainsi, une règle MQTT est toujours clonée, comme le montre la ligne 33 du listing 3.2, en fin d'exécution pour rester dans la file de règles que le moteur exécutera à la prochaine itération. De plus, si la règle n'a reçu aucun message, elle ne sera pas activée et uniquement la méthode de génération de règle sera exécutée.

Pour utiliser ce type de règle, il suffit dès lors de créer une nouvelle classe étendant la classe abstraite `AbstractMqttRule`. Lors de sa création, la règle sera automatiquement souscrite (en utilisant la méthode `subscribeRule` comme le montre la ligne 12) au filtre de topics passé en paramètre du constructeur de la classe.

Listing 3.2 – Classe abstraite représentant une règle de type MQTT.

```
1  public abstract class AbstractMqttRule extends AbstractRule {
2
3      private List<MqttMessage> messages;
4      private String topicFilter;
5
6
7      public AbstractMqttRule(GenericProactiveEngine engine,
8                              String topicFilter) {
9          this.engine = engine;
10         this.messages = new ArrayList<>();
11         this.topicFilter = topicFilter;
12
13         this.engine.subscribeRule(this);
14     }
15
16     @Override
17     public final boolean execute() {
18         if (!messages.isEmpty()) {
19             dataAcquisition();
20             if (activationGuards()) {
21                 setActivated(true);
22                 if (conditions()) {
23                     actions();
24                 }
25             } else {
26                 setActivated(false);
27             }
28         } else {
29             setActivated(false);
```

```
30          }
31
32          boolean r = rulesGeneration();
33          createRule(this);
34
35          return r;
36      }
37
38      public String getTopicFilter() {
39          return topicFilter;
40      }
41
42      public void setTopicFilter(String topicFilter) {
43          this.topicFilter = topicFilter;
44      }
45
46      public List<MqttMessage> getMessages() {
47          return messages;
48      }
49
50  }
```

3.4.3.2 Fonctionnement

Avec les différentes modifications apportées au moteur proactif, il est maintenant possible d'écrire des scénarios permettant la récolte d'informations sur un environnement équipé de capteurs, mais aussi d'envoyer des commandes aux actionneurs de ce même environnement. La figure 3.7 résume le mode de fonctionnement entre le moteur proactif et le système IoT. Cette figure reprend deux composants de l'architecture principale de la solution proposée à la figure 3.2 : le moteur proactf et le système IoT. Le moteur proactif contient la file de règles à exécuter mais s'exécute aussi conjointement avec le broker MQTT, tandis que le système IoT contient divers capteurs et actionneurs.

L'idée est que les deux types de règles coexistent au sein de la file du moteur proactif, à savoir les règles « normales » et celles de type « MQTT ». Une règle MQTT serait souscrite à un filtre de topic, leur permettant de recevoir les informations remontées à partir des capteurs publiant sur divers topic. Par exemple, sur la figure 3.7, la règle 2 est abonnée au filtre de topic `/filtre/topic/1/+`, si le capteur 1 publie sur le topic `/filtre/topic/1/capteur1` et le capteur 2 publie sur le topic `/filtre/topic/1/capteur2`, la règle 2 recevra tous les messages des capteurs 1 et 2.

Le rôle du broker est tel que défini par les spécifications du protocole MQTT. Il sert d'intermédiaire entre les clients qui publient des messages et les clients qui reçoivent des messages car ils ont souscrit à un ou plusieurs topics [34]. Un client peut aussi jouer les deux rôles, comme déjà expliqué, une règle peut être souscrite à un ou plusieurs topics pour recevoir des données, mais elle peut aussi envoyer

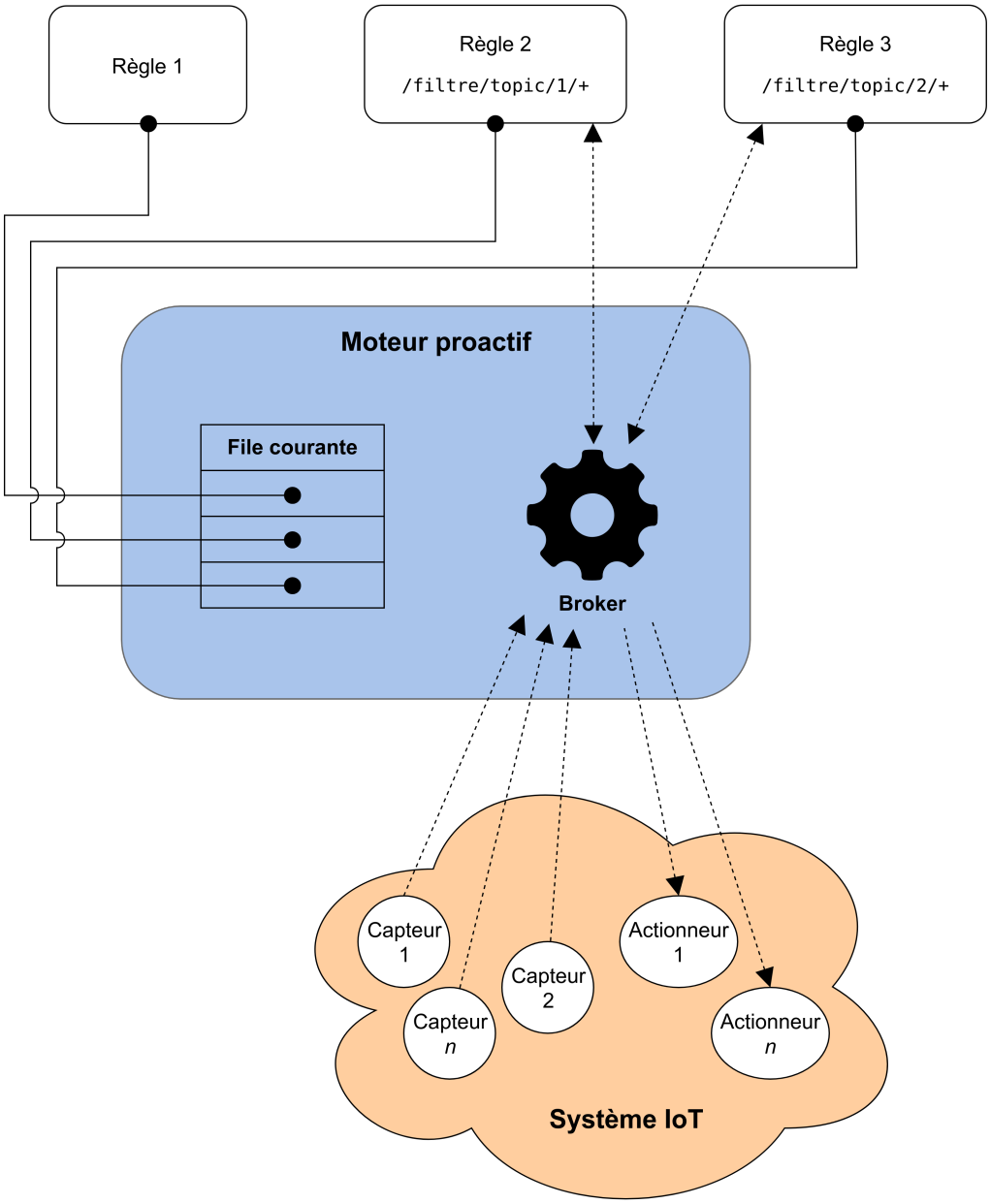

Figure 3.7 – Détail du fonctionnement du moteur proactif avec un système IoT.

des messages via le broker sur un certain topic. Notons que n'importe quel type de règle peut utiliser le broker pour envoyer des messages aux actionneurs, puisque que chacun règle possède une référence vers le moteur qui l'exécute et donc une référence vers le broker qui propose la méthode `sendMessage`. Les capteurs joueront uniquement le rôle de clients qui publient des messages sur un topic, alors que les actionneurs seront ceux qui reçoivent des messages en souscrivant à un topic.

Pour terminer, l'avantage principal de cette manière de fonctionner est qu'il est possible de créer des scénarios avec comme règle de déclenchement, une règle de type MQTT. Ensuite, cette règle pourrait créer une ou plusieurs autres règles auxquelles

les messages reçus seraient passés en paramètre. Ces autres règles pourraient alors réaliser différents calculs pour traiter, agréger, combiner, etc. les données reçues, et ainsi faire ressortir des « méta »-données, ou comme décrit à la section 1.3.3, des informations classifiées comme appartenant au contexte secondaire. Enfin, ces données relatant du contexte secondaire pourraient être stockées d'une certaine manière par ces mêmes règles ou par d'autres, pour être ensuite utilisées dans les politiques de sécurité. La façon dont ce stockage est réalisé et comment les données sauvegardées sont référencées est le sujet de la section suivante. De plus, une ou plusieurs règles de ce scénario pourraient envoyer des messages aux actionneurs, lorsque cela est nécessaire, afin d'agir sur l'environnement.

3.5 Référencement des attributs dans les politiques de contrôle d'accès

Dans la section précédente, il était question de la communication entre le moteur proactif et le système IoT. L'autre aspect important qu'il reste à détailler, est comment stocker des données pour qu'elles soient accessibles au PDP, tel que décrit dans la section 3.3, mais aussi comment les référencer dans les politiques lorsque ces données sont effectivement stockées.

Cette section finit de répondre à la première question de la problématique posée au chapitre 2, c'est-à-dire la façon d'alimenter de modèle ABAC avec des données provenant du système IoT, qui avait été abordée en partie à la section précédente. En effet, nous savons maintenant comment récupérer les données des capteurs grâce aux capteurs, mais il reste encore à faire le lien entre le moteur et le modèle ABAC.

À ce niveau, les possibilités pour la façon de stocker les données (du contexte secondaire) sont plus réduites puisque utiliser une base de données semble être un choix trivial, d'autant plus que le moteur offre déjà la possibilité de facilement prendre en compte une nouvelle base de données avec laquelle le moteur devrait interagir. Le choix se porterait donc sur l'utilisation d'une base de données relationnelle ou non-relationnelle. Il s'avère que le moteur a été conçu pour principalement fonctionner avec une base de données relationnelles, et en particulier MySQL. C'est pourquoi le choix dans ce cas-ci s'est également tourné vers MySQL, dans un souci de faciliter et pour ne pas modifier le fonctionnement existant du moteur.

3.5.1 Base de données et moteur proactif

Le moteur propose différentes abstractions afin de prendre en charge les interactions avec une base de données MySQL. Il s'agit plus concrètement de *wrappers*, avec comme première couche d'abstraction la classe Java `AbstractDataAccessWrapper` qui prend la forme d'un singleton et qui a pour but de regrouper les paramètres communs à une connexion à une base de données. La deuxième couche d'abstraction est réalisée à travers la classe `AbstractHostDataAccessWrapper` qui étend la classe `AbstractDataAccessWrapper`. Celle-ci permet de différencier le comportement d'un *wrapper* communiquant avec une base de données « hôte », du comportement d'un *wrapper* (`AbstractEngineDataAccessWrapper`) communiquant avec la base de données interne au moteur proactif.

Ainsi, pour ajouter la possibilité de communication avec une base de données (MySQL) il suffit d'implémenter un *wrapper* concret qui étend la classe `AbstractHostDataAccessWrapper`. Puisque ce *wrapper* sera référencé au sein du moteur proactif, chaque règle pourra l'utiliser pour insérer, supprimer, modifier, etc. les données de la base de données hôte. Bien sûr, il sera nécessaire que le développeur écrive les différentes méthodes permettant d'exécuter les requêtes SQL correspondantes aux actions nécessaires. De plus, le schéma de la base de données relationnelle dépendra fortement du type de données utilisées dans les politiques de sécurité, mais aussi de types de capteurs proposés par le système IoT.

3.5.2 Utiliser une base de données comme source du PIP

Dans le cas de la solution proposée, l'idée est donc d'utiliser une base de données pour alimenter le PIP en données provenant (indirectement lorsqu'elles ont été traitées par un scénario) des capteurs du système IoT. Nous savons déjà qu'il est assez facilement possible d'interagir avec une base de données à travers les règles du moteur proactif. Maintenant, comme le définit par le standard XACML, c'est le PIP qui va devoir jouer le rôle d'intermédiaire entre le PDP et les valeurs d'attributs stockées dans la base de données dont il est question.

Ici, tout dépend de l'implémentation de XACML utilisée. En effet, c'est le futur système XACML (ou celui déjà en place) qui devra prendre en compte dans son PIP la même base de données que celle utilisée par le moteur proactif pour stocker les valeurs des attributs. La solution ne contraint volontairement pas l'implémentation XACML utilisée dans le but de rester général et de pouvoir s'implanter dans des systèmes existants. Il s'agit aussi de ne pas être dépendant par rapport à une certaine implémentation XACML. Cependant, il est possible de décrire comment le PIP devrait normalement fonctionner.

3.5.3 Rôle du PIP et attributs « externes »

Comme déjà abordé dans le cadre de ce travail, à la section 3.3.1.2 et à la figure 3.3, le PIP sert de point d'ancrage à l'architecture ABAC pour le moteur proactif et ce indirectement via la base de données communes.

Mais d'un point de vue ABAC, le PIP sert surtout à récupérer des valeurs d'attributs manquantes pour que le PDP puissent évaluer correctement une politique. Pour mieux comprendre ce fonctionnement, mais aussi comment les valeurs d'attributs stockées dans la base données peuvent être référencés, prenons un simple exemple. Posons la politique définie au listing 3.3 comme applicable à une requête hypothétique reçue par le PDP (la valeur de la balise `Target` de la politique du listing 3.3 a été omise par simplicité). Si la requête ne contient pas la valeur de l'attribut désigné par la balise `AttributeDesignator` qui référence une valeur de température dans ce cas, le PDP fera appel au PIP, en lui passant en paramètres les informations nécessaires (comme la requête XACML courante et l'`AttributeDesignator` dont il est question), pour récupérer cette valeur manquante.

Listing 3.3 – Exemple d'une politique XACML.

```
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
  PolicyId="ExamplePolicy"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
  ↪ algorithm:first-applicable"
  Version="1.0">
  <Target>
  </Target>
  <Rule Effect="Permit" RuleId="Rule_1">
    <Target>
    </Target>
   <Condition>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
      ↪ double-greater-than">
        <AttributeValue
        ↪ DataType="http://www.w3.org/2001/XMLSchema#double">27.0
        ↪ </AttributeValue>
        <AttributeDesignator
            AttributeId="http://foo.bar/sensors/temperature"
            Category="urn:oasis:names:tc:xacml:1.0:subject-category:
            ↪ access-subject"
            DataType="http://www.w3.org/2001/XMLSchema#double"
            MustBePresent="true"/>
      </Apply>
    </Condition>
  </Rule>
  <Rule Effect="Deny" RuleId="Deny-Rule"/>
</Policy>
```

Dès lors, le PIP, en interne, va interroger une source externe d'attributs (en fonction de comment il a été implémenté) et renvoyer la valeur récupérée au PDP. Dans notre cas, pour récupérer cette valeur le PIP peut interroger la base de données contenant les valeurs des attributs que les scénarios du moteur proactif ont inséré. Par exemple, une simple requête SQL comme celle qui suit peut servir à récupérer la valeur de la température référencée par l'identifiant "`http://foo.bar/sensors/temperature`" :

```
SELECT temperature FROM attributes_values
```

En supposant que la table `attributes_values` contient toujours une seule ligne contenant la dernière valeur de température insérée.

Pour terminer, la réponse à la deuxième question est maintenant complète, puisque nous savons qu'il est tout à fait possible d'écrire des scénarios où des règles peuvent stocker dans une base de données des informations brutes ou du contexte secondaire à partir de celles remontées par les capteurs. De plus, il a aussi été montré que cette base de données pourra être accédée par un PIP pour que les valeurs qui s'y trouvent puissent être utilisées lors de l'évaluation de politiques. Le prochain

chapitre va permettre de concrétiser les concepts peut-être encore abstraits exposés à travers les sections précédentes.

Chapitre 4

Preuve de concept

Ce chapitre permet de concrétiser la solution élaborée au chapitre 3, et ce au travers d'un exemple. Avant tout, il est nécessaire de définir, à la section 4.1, le contexte dans lequel l'exemple est réalisé. Ensuite, la section 4.2 démontre comment il est possible d'utiliser le moteur proactif en définissant un scénario (c.-à-d. un ensemble coordonné de règles) reflétant les possibilités qu'offre la combinaison entre un moteur proactif, le modèle ABAC et un système IoT. La section 4.3 détaille comment référencer les attributs externes dans les politiques XACML. Enfin, la section 4.4 conclut ce chapitre par une évaluation.

4.1 Contexte

Pour réaliser cette preuve de concept, il aurait été possible d'utiliser des capteurs et actionneurs réels, en réalisant un « banc de test » miniature. Cependant, il a été choisi de simuler ces capteurs et actionneurs, principalement par simplicité, car il est plus rapide et plus simple de les simuler de manière logicielle que de se les procurer et de les configurer. Ainsi, un cas d'utilisation fictif a été défini, donnant ainsi un cadre pragmatique pour réaliser une preuve de concept.

4.1.1 Centre de données modulaire

Le cas d'utilisation dont il est question dans cette section est celui d'un centre de données modulaire. L'idée générale est la suivante : le centre de données est composé de plusieurs modules, où chacun des modules contient tout le matériel nécessaire pour faire fonctionner des serveurs disposés sur des racks. Cela concerne donc, en plus des serveurs, une alimentation en électricité (souvent complétée avec un bloc d'alimentation d'urgence), les équipements réseau, un système de refroidissement, un système de détection, d'alerte, et d'extinction d'incendie, ainsi que différents dispositifs de sécurité. Un tel centre de données peut donc contenir un nombre maximum de modules, qui peuvent être ajoutés et enlevés « à la volée ».

Chaque module contient également des capteurs et actionneurs, positionnés aux endroits appropriés. Par exemple, les capteurs de flux d'air seront placés en entrée

et en sortie de flux d'air des racks. Pour l'exemple de scénario de la section 4.2, les capteurs et actionneurs nécessaires sont les suivants :

— *Capteurs de température* : Pour s'assurer de l'état optimal des serveurs présents dans un module, il est nécessaire de surveiller la température. En effet, pour fonctionner le plus efficacement possible, des appareils électroniques tels que des serveurs doivent être maintenus à une certaine température, mais ceux-ci dégagent beaucoup de chaleur, c'est pourquoi il est nécessaire de positionner des capteurs à différents endroits à travers les racks. De plus, les valeurs de température pourraient être utilisées dans des politiques XACML, permettant ainsi la prise de décision en fonction de l'état des serveurs.

— *Système de refroidissement* : Ce système joue le rôle d'actionneur et permet donc de réguler la température interne d'un module, et donc des serveurs. Il peut aussi être utilisé pour maintenir une humidité idéale au sein d'un module.

Bien sûr, un module peut être équipé d'un bon nombre d'autres capteurs et actionneurs. Cependant, il s'agit d'un cas d'application fictif (dans le sens où des hypothèses simplificatrices sont établies), se rapprochant le plus possible d'un environnement réel, permettant de donner une idée des capacités et des avantages (ou désavantages) que fournit la solution principale proposée dans ce document.

4.2 Scénario

4.2.1 Description générale

Ce scénario vise, d'une part à stocker de manière cohérente une valeur de température « agrégée » dans la base de données liée au PIP, et d'autre part à prendre les actions nécessaires sur l'environnement via les actionneurs (uniquement le système de refroidissement dans ce cas) d'un module. La figure 4.1 représente la structure suivie par ce scénario. Il se déclenche lorsqu'au moins un message a été reçu par la règle initiale (4.2.2.1 `SENSORS_TEMPERATURE_MONITORING`). Ensuite, pour stocker une valeur représentant la température « globale » dans la base de données liée au PIP, afin que les politiques XACML puissent l'utiliser, la règle se base sur le nombre de messages reçus (depuis la dernière itération) pour choisir la manière dont ceux-ci doivent être traités.

Si la liste ne contient qu'un seul message, nous stockons directement la valeur qu'il contient, en vérifiant qu'elle n'est pas anormalement trop éloignée de celle déjà stockée dans la base de données. Par contre, si elle en contient deux (4.2.2.2 `TEMPERATURE_COMPARE`), les deux valeurs sont comparées, si celles-ci ne sont pas anormalement éloignées, la moyenne entre les deux est stockée. De plus, si la liste contient plus de 2 messages (4.2.2.3 `TEMPERATURE_MAD`), nous nous cantonnons tout de même aux 20 derniers reçus, dans le but d'analyser uniquement les plus récents. Dans ce cas, pour détecter les valeurs anormales dans l'échantillon, nous utilisons une mesure statistique, appelée Median absolute deviation (MAD). Toujours dans ce même cas, nous stockerons dans le PIP la médiane des valeurs de température, et non pas la moyenne, pour ne pas être influencé par les possibles

valeurs anormales (par exemple, dues à un dysfonctionnement d'un des capteurs) ne reflétant peut-être pas l'état réel de la température à l'intérieur d'un module.

Lors du stockage d'une valeur de température dans le PIP (4.2.2.4 `TEMPERATURE_PIP`), il est nécessaire d'adapter le système de refroidissement en fonction de celle-ci (4.2.2.5 `ACTUATOR_ADJUST_AIR_COOLING`). Enfin, lorsqu'une valeur anormale de température est détectée, un compteur est incrémenté dans la base de données interne au moteur. Lorsque celui-ci dépasse un certain nombre, une règle, toujours présente dans la queue du moteur, est activée afin de déclencher une alerte (4.2.2.6 `TEMPERATURE_ABNORMAL_VALUES_ALERT`).

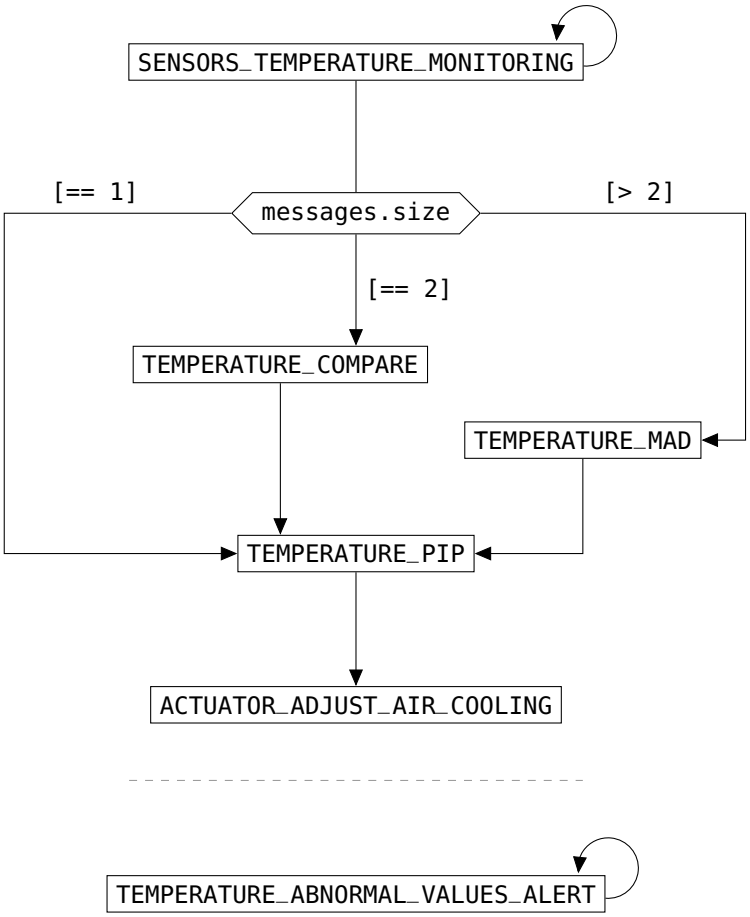

FIGURE 4.1 – Structure du scénario d'exemple.

4.2.2 Implémentation

En remarque préliminaire, notons que chacune des règles décrites ci-après possède comme paramètre l'identifiant du module auquel la règle s'applique, défini par la variable `moduleID`. Ceci permettra de créer les règles autant de fois qu'il existe de module au sein du centre de données.

4.2.2.1 Règle `SENSORS_TEMPERATURE_MONITORING`

Paramètres

/

Description

C'est la règle d'initialisation du scénario. Elle est souscrite au filtre de topic suivant : `<moduleID>/sensors/temperature/<sensorID>`, en admettant que les capteurs de température publient leurs valeurs sur des topics respectant la forme définie par ce filtre, où `<sensorID>` varie en fonction du capteur et `<moduleID>` varie en fonction du module dans lequel les capteurs sont placés. Elle reçoit donc les messages MQTT postés sur ces deux topics, et elle se charge ensuite de les passer à l'une des trois règles suivantes : `TEMPERATURE_PIP`, `TEMPERATURE_COMPARE` ou `TEMPERATURE_MAD`, en fonction de la taille de la liste des messages reçus. Sa garde d'activation a pour but de ne pas exécuter la règle si le module n'est pas présent dans le centre de données.

De plus, afin de ne pas surcharger les règles, et plus particulièrement la règle `TEMPERATURE_MAD`, uniquement les `N_TEMPERATURES_TO_TAKE` derniers messages sont utilisés et les autres sont « oubliés ». Ainsi, nous nous assurons de traiter les plus récents messages sans vraiment perdre trop d'information. La valeur de `N_TEMPERATURES_TO_TAKE` dépend de la fréquence d'envoi des capteurs.

Pseudo code Java

```
data acquisition : /

activation guards :
  return getPIP().isModuleAttached();

conditions :
  return true;

actions :
  if (messages.size() > N_TEMPERATURES_TO_TAKE) {
    messages.subList(0,
      (messages.size() - 1) - N_TEMPERATURES_TO_TAKE).clear();
  }

  temperaturesToAnalyze = new ArrayList<>(messages);

rules generation :
  if (getActivated()) {
    switch (temperaturesToAnalyze.size()) {
        case 1:
          createRule(new
          ↪  TEMPERATURE_PIP(temperaturesToAnalyze.get(0)));
```

```
22            break;
23          case 2:
24            createRule(new TEMPERATURE_COMPARE(temperaturesToAnalyze));
25            break;
26          default:
27            createRule(new TEMPERATURE_MAD(temperaturesToAnalyze));
28            break;
29      }
30    }
31    return true;
```

4.2.2.2 Règle `TEMPERATURE_COMPARE`

Paramètres

— `temperatureMessages` : la liste des messages MQTT (qui contient exactement 2 éléments) contenant les valeurs de température.

Description

Cette règle prend en charge le cas simple où la liste des messages reçus contient seulement 2 messages et où la règle `TEMPERATURE_MAD` ne peut pas s'appliquer. Pour ce faire, elle compare les deux valeurs et vérifie si l'écart entre celles-ci n'est pas trop élevé (selon une valeur arbitrairement choisie, définie par la constante `ABNORMAL_THRESHOLD`, de préférence assez haute puisqu'il est possible que ces deux valeurs viennent de capteurs différents). Si l'écart est raisonnable, elle crée alors la règle `TEMPERATURE_PIP` pour stocker la moyenne entre les deux valeurs. Sinon, dans le cas où l'écart serait aberrant, le compteur de valeurs anormales est incrémenté.

Pseudo code Java

```
1  data acquisition : /
2
3  activation guards :
4    return temperatureMessages.size() == 2;
5
6  conditions :
7    return true;
8
9  actions :
10   if (Math.abs(temperatureMessages.get(0) -
     ↪  temperatureMessages.get(1)) < ABNORMAL_THRESHOLD) {
11     abnormalDetected = false;
12     temperatureToStore = (temperatureMessages.get(0) +
       ↪  temperatureMessages.get(1)) / 2;
13   } else {
14     abnormalDetected = true;
15     getEngineDB().incrementAbnormalTemperatureValuesCounter();
```

```
16    }
17
18  rules generation :
19    if (getActivated() && !abnormalDetected) {
20      createRule(new TEMPERATURE_PIP(temperatureToStore));
21    }
22    return true;
```

4.2.2.3 Règle `TEMPERATURE_MAD`

Paramètres

— `temperatureMessages` : la liste des messages MQTT (dont la taille est supérieure à 2) contenant les valeurs de température.

Description

Cette règle vérifie qu'aucune valeur n'est anormale, dans le sens où elle serait trop élevée ou trop basse par rapport aux autres valeurs présentes dans la liste `temperatureMessages`. L'idée est donc d'utiliser une mesure statistique appelée *Median absolute deviation (MAD)*. Celle-ci permet de calculer la distance moyenne d'une valeur à la médiane. Ainsi, si une valeur de température est éloignée de la médiane de plus d'un certain nombre de fois la mesure *MAD*, elle devra être considérée comme anormale.

Par exemple, prenons l'échantillon $X = (23, 24, 25, 24, 26, 40, 23, 22, 40)$ (valeurs de température en degré Celsius) dont la médiane est, $median(X) = 24$. Il est possible de calculer l'écart absolu des valeurs par rapport à cette médiane, $|X_i - median(X)| = (1, 0, 1, 0, 2, \mathbf{16}, 1, 2, \mathbf{16})$. Enfin, $MAD = median(|X_i - median(X)|) = 1$. S'il est établi au préalable que toutes les valeurs éloignées de plus de 3 fois la *MAD* doivent être considérées comme anormales, nous voyons clairement que les valeurs 40 de l'échantillon X, seront bien considérées comme anormales (leur écart absolu à la médiane est de 16).

Lorsqu'une telle valeur anormale est détectée, il est possible d'incrémenter en base de données (interne au moteur) un compteur de valeurs anormales détectées. Lorsqu'un ce compteur dépasse un certain nombre, il est établi qu'un problème est survenu et la valeur de température stockée dans la base de données liée au PIP est donc redéfinie à une valeur explicitement élevée (scénario le plus négatif), ce qui est le rôle de la règle `TEMPERATURE_ABNORMAL_VALUES_ALERT` (4.2.2.6). Enfin, c'est la valeur médiane qui est stockée (en créant la règle `TEMPERATURE_PIP`), qui permet d'éviter une trop grande influence des valeurs extrêmes.

Pseudo code Java

```
1  data acquisition : /
2
3  activation guards :
4    return temperatureMessages.size() > 2;
```

```
5
6  conditions :
7    return true;
8
9  actions :
10   double median = getMedian(temperatureMessages);
11   double mad = getMAD(temperatureMessages, median);
12   for (MqttMessage tempMessage : temperatureMessages) {
13     double tmp = tempMessage.getValue();
14     if (Math.abs(tmp - median) > (3 * mad)) {
15       getEngineDB().incrementAbnormalTemperatureValuesCounter();
16       break;
17     }
18   }
19
20   temperatureToStore = median;
21
22 rules generation :
23   if (getActivated()) {
24     createRule(new TEMPERATURE_PIP(temperatureToStore));
25   }
26   return false;
```

4.2.2.4 Règle `TEMPERATURE_PIP`

Paramètres

— `temperature` : la valeur de la température à stocker dans le PIP.

Description

Cette règle vérifie dans un premier temps s'il est nécessaire de stocker dans le PIP la valeur passée en paramètre. Cette vérification équivaut à comparer la valeur déjà présente dans le PIP et la nouvelle. Si la différence est supérieure ou égale à `MINIMUM_DELTA`, alors la nouvelle valeur est stockée.

Si la règle est activée, il est également vérifié que, si le temps écoulé depuis la dernière mise à jour de la valeur de température en base de données est inférieur à un certain intervalle (défini par `INTERVAL`), la valeur qui doit être stockée n'est pas anormalement élevée par rapport à celle déjà présente dans le PIP. En effet, la température n'a normalement pas tendance à augmenter brutalement dans le temps, sauf lors de problème majeur, auquel cas le compteur de valeurs de températures anormales est incrémenté.

Enfin, cette règle génère la règle `ACTUATOR_ADJUST_AIR_COOLING`, en lui passant en paramètre la température qui devait être stockée, afin de réguler le système de refroidissement.

Pseudo code Java

```
data acquisition :
  temperatureInPIP = getPIP().getTemperature();
  lastTempUpdate = getPIP().getLastTemperatureUpdate();

activation guards :
  return Math.abs(temperatureInPIP - temperature) > MINIMUM_DELTA;

conditions :
  return true;

actions :
  if ((lastTempUpdate.getTime() + INTERVAL) >
  ↪  System.currentTimeMillis()
    && Math.abs(temperature - temperatureInPIP) > MAXIMUM_DELTA) {
    getEngineDB()
    ↪  .incrementAbnormalTemperatureValuesCounter(moduleID);
  } else {
    getPIP().updateTemperature(moduleID, temperature);
  }

rules generation :
  createRule(new ACTUATOR_ADJUST_AIR_COOLING(moduleID, temperature))
  return true;
```

4.2.2.5 Règle `ACTUATOR_ADJUST_AIR_COOLING`

Paramètres

— `temperature` : la température à prendre en compte pour l'ajustement.

Description

Cette règle vise à ajuster le système de refroidissement en fonction de la température. Pour y arriver, il est nécessaire de définir deux niveaux : une valeur de température considérée comme borne minimum (`LIMIT_MIN`) et une autre comme borne maximum (`LIMIT_MAX`). Par exemple, pour un fonctionnement optimal du matériel, ces bornes peuvent être respectivement égales à 18 °C et 27 °C. Ainsi, cette règle compare où se situe la valeur courante de la température et prend l'action nécessaire selon les cas suivants :

— Si la valeur est inférieure ou égale à la borne minimum, il faut placer le système de refroidissement au minimum.

— Si la valeur est entre les deux bornes, pour essayer d'être quelque peu préventif, il est possible vérifier à quel point la valeur de température se rapproche d'une des deux bornes. C'est pourquoi, lorsque $LIMIT_{min} < temperature_{current} < LIMIT_{max}$, si la valeur $abs(temperature_{current} -$

$LIMIT_{min}$) ou la valeur $abs(temperature_{current} - LIMIT_{max})$ est inférieure ou égale à 2, le système de refroidissement est ajusté en fonction de la borne.

— De manière similaire au premier point, si la température est supérieure ou égale à la borne maximum, il faut mettre le système de refroidissement au maximum.

Remarquons que la méthode `setAirCoolingLevel(...)` s'occupe de l'interaction avec l'actionneur lié au système de refroidissement, en envoyant un message MQTT via le broker sur un topic prédéfini.

Pseudo code Java

```
data acquisition : /

activation guards :
  return true;

conditions :
  return true;

actions :
  if (temperature < LIMIT_MIN) {
    setAirCoolingLevel(AirCoolingLevel.LOW);
  } else if (temperature > LIMIT_MAX) {
    setAirCoolingLevel(AirCoolingLevel.HIGH);
  } else {
    if ((temperature - LIMIT_MIN) <= 2) {
      setAirCoolingLevel(AirCoolingLevel.LOW);
    } else if (Math.abs(temperature - LIMIT_MAX) <= 2) {
      setAirCoolingLevel(AirCoolingLevel.HIGH);
    } else {
      setAirCoolingLevel(AirCoolingLevel.MODERATE);
    }
  }

rules generation :
  return false;
```

4.2.2.6 Règle `TEMPERATURE_ABNORMAL_VALUES_ALERT`

Paramètres

/

Description

Cette règle a pour but de détecter quand il est nécessaire de déclencher une alerte lorsque le compteur de valeurs anormales (stocké dans la base de données

interne au moteur proactif) dépasse une certaine valeur. Elle se réplique à chaque itération pour constamment contrôler ce compteur.

Pseudo code Java

```
1  data acquisition :
2    abnormalValuesCounter =
     ↪  getEngineDB().getAbnormalTemperatureValuesCounter();
3
4  activation guards :
5    return getPIP().isModuleAttached();
6
7  conditions :
8    return abnormalValuesCounter > 3;
9
10 actions :
11   generateAlert("Abnormal temperature values detected in module " +
     ↪  moduleID);
12   getEngineDB().resetAbnormalTemperatureValuesCounter();
13
14 rules generation :
15   createRule(this);
16   return true;
```

4.3 Politique XACML et attributs externes

Dans la section 4.2, il a été démontré, à travers un exemple, comment il est possible d'utiliser le moteur proactif et le concept de scénario pour récupérer des données provenant de capteurs, stocker à partir des celles-ci des informations du contexte secondaire en base de données et agir sur l'environnement grâce aux actionneurs lorsque c'est nécessaire. Cependant, afin de clôturer cet exemple, cette section décrit le côté « contrôle d'accès » de la solution.

Nous savons qu'une valeur de température « agrégée » est présente dans la base de données liée au PIP. Posons la politique définie au listing 4.1 et admettons aussi qu'un composant chargé de la répartition de charge (entre les modules) est présent dans le centre de données modulaire décrit précédemment. Ainsi, le but de la politique est de permettre au répartiteur de charge de savoir si un module est disponible pour prendre en charge une requête à destination d'un des serveurs présents dans ce module. Pour ce faire, il peut s'appuyer sur un PEP qui interceptera la requête voulant accéder à un serveur interne d'un module, et qui transformera cette requête en une requête de type XACML telle que celle décrite au listing 4.2 (où le module qui souhaite être accédé porte l'identifiant `module01` dans ce cas). La politique définie au listing 4.1 s'y appliquera puisque la balise `Target` définit que cette politique vise les requêtes contenant un attribut référençant le sujet dont la valeur doit être `load-balancer`, qui est bien présent à la ligne 9 de la requête XACML du listing 4.2.

L'intérêt principal de cette politique est de montrer la possibilité de référencer des attributs « externes », comme le montrent les lignes 24 à 34 du listing 4.1. C'est plus précisément la balise `AttributeDesignator` avec l'attribut `AttributeId="data-center-temperature"` qui le permet. En effet, comme expliqué à la section 3.5, le PDP pourra utiliser le PIP pour récupérer cette valeur en base de données, qui y aura été stockée par un scénario du moteur proactif. En l'occurrence, cette politique bloque les accès dans le cas où la valeur de la température dépasse 27,0 et dans les autres cas, l'accès est autorisé.

Listing 4.1 – Politique utilisant la température interne d'un module.

```
1  <Policy PolicyId="module-temperature"
2    RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-
     ↪ algorithm:deny-overrides"
3    Version="1.0"
4    xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
5
6    <Target>
7      <AnyOf>
8        <AllOf>
9          <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:
           ↪ string-equal">
10           <AttributeValue
             ↪ DataType="http://www.w3.org/2001/XMLSchema#string">
11             load-balancer
12           </AttributeValue>
13           <AttributeDesignator
14             MustBePresent="True"
15             Category="urn:oasis:names:tc:xacml:1.0:subject-category:
               ↪ access-subject"
16             AttributeId="urn:oasis:names:tc:xacml:1.0:subject:
               ↪ subject-id"
17             DataType="http://www.w3.org/2001/XMLSchema#string"/>
18         </Match>
19       </AllOf>
20     </AnyOf>
21   </Target>
22
23   <Rule Effect="Deny" RuleId="less-than-max-temperature">
24     <Condition>
25       <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
         ↪ double-greater-than">
26         <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
           ↪ double-one-and-only">
27           <AttributeDesignator AttributeId="data-center-temperature"
28             Category="urn:oasis:names:tc:xacml:3.0:attribute-
               ↪ category:environment"
29             DataType="http://www.w3.org/2001/XMLSchema#double"
```

```
            MustBePresent="true"/>
        </Apply>
        <AttributeValue
        ↪ DataType="http://www.w3.org/2001/XMLSchema#double">27.0
        ↪ </AttributeValue>
      </Apply>
    </Condition>
    <AdviceExpressions>
      <AdviceExpression
      ↪ AdviceId="deny-data-center-temperature-advice"
      ↪ AppliesTo="Deny">
        <AttributeAssignmentExpression AttributeId="urn:oasis:names:
        ↪ tc:xacml:2.0:example:attribute:text">
          <AttributeValue
          ↪ DataType="http://www.w3.org/2001/XMLSchema#string">
            The temperature of the module is too high to allow
            ↪ access.
          </AttributeValue>
        </AttributeAssignmentExpression>
      </AdviceExpression>
    </AdviceExpressions>
  </Rule>

  <Rule Effect="Permit" RuleId="default-case"/>

</Policy>
```

Listing 4.2 – Requête XACML à destination du PDP.

```
<Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
↪ CombinedDecision="false" ReturnPolicyIdList="false">
    <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-
    ↪ category:action">
        <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:
        ↪ action-id"
        ↪ IncludeInResult="false">
            <AttributeValue
            ↪ DataType="http://www.w3.org/2001/XMLSchema#string">
            ↪ access</AttributeValue>
        </Attribute>
    </Attributes>
    <Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-
    ↪ category:access-subject">
        <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:
        ↪ subject:subject-id"
        ↪ IncludeInResult="false">
```

```
9             <AttributeValue
              ↪ DataType="http://www.w3.org/2001/XMLSchema#string">
              ↪ load-balancer</AttributeValue>
10        </Attribute>
11    </Attributes>
12    <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-
      ↪ category:resource">
13        <Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:
          ↪ resource:resource-id"
          ↪ IncludeInResult="true">
14            <AttributeValue
              ↪ DataType="http://www.w3.org/2001/XMLSchema#string">
              ↪ module01</AttributeValue>
15        </Attribute>
16    </Attributes>
17 </Request>
```

4.4 Évaluation

Cette preuve de concept à travers un exemple décrit dans les sections précédentes aborde la réponse aux deux questions de la problématique du chapitre 2. En effet, le scénario de la section 4.2 et plus particulièrement la règle de la section 4.2.2.4 nous montre la façon dont le moteur proactif est utilisé pour stocker des données dans la base de données liée au PIP, répondant ainsi à la première sous-question de la problématique. Remarquons que l'avantage principal est que ces données ne sont pas les données brutes provenant directement des capteurs, mais qu'elles peuvent être prétraitées et donc être de plus haut-niveau, comme celles du contexte secondaire. Ceci permet donc d'utiliser des attributs correspondants à des caractéristiques de l'environnement, sans être toutefois accablé par les données brutes, souvent en très grands nombres et pas toujours fiables.

Le second avantage réside dans la réponse à la deuxième sous-question de la problématique et prend son sens dans la règle de la section 4.2.2.5. Cette dernière nous montre que les actionneurs peuvent être utilisés spontanément par le scénario, dans le but d'agir sur le monde physique. Ainsi, il est possible de réagir le plus rapidement possible, par exemple lorsque des failles de sécurité sont détectées au vu des données qui ont été récupérées.

4.4.1 Aspect générique

Une caractéristique importante de la solution est son aspect générique. En effet, une certaine liberté est laissée au programmeur de scénario lors de l'élaboration d'un système. Bien sûr, il devra respecter la structure des règles telles qu'elles sont définies dans le contexte du moteur proactif. Néanmoins, c'est à lui que revient la tâche d'écrire et d'agencer ces règles. C'est aussi là l'aspect original et intéressant, puisque nous proposons d'utiliser un système à base de règles, assez générique pour

être utilisé dans des situations variées, évitant ainsi la réécriture de nouveau système ad hoc pour chacune de ces situations.

De plus, la division en scénarios et la capacité du moteur proactif à en exécuter plusieurs en parallèle montrent également qu'il est possible de gérer plusieurs capteurs et actionneurs. Cette même division fournit aussi au développeur ce qu'on pourrait appeler un framework permettant de définir des scénarios se concentrant sur un seul type de données ou d'actionneurs à la fois. L'aspect « séparation des préoccupations » est donc ici une caractéristique supplémentaire, simplifiant le développement et la maintenance, puisque chaque scénario peut être amené à évoluer de manière indépendante.

La principale contrainte reste la façon dont la communication avec les capteurs et actionneurs est établie, à savoir avec le protocole MQTT. Cependant, il semble envisageable de la contourner, lorsque cela est vraiment nécessaire, avec un gateway faisant le pont entre notre système basé sur MQTT et tout autre protocole ou technologie souhaitant être utilisé, par exemple dans le cas d'une implémentation où le système IoT est déjà en place. Il existe également la possibilité de directement étendre les fonctionnalités du moteur proactif, à l'image de ce qui a été fait avec le protocole MQTT.

4.4.2 Scalabilité

Comme énoncé précédemment, le moteur est capable de supporter plusieurs appareils IoT, dès lors la question de la scalabilité peut se poser. Plusieurs facteurs sont importants pour y répondre.

Premièrement, le nombre de capteurs et d'actionneurs à gérer pourrait faire augmenter le nombre de scénarios que le moteur proactif devrait exécuter. Comme il existe un nombre de règles maximum qu'il est possible d'exécuter en une seule itération, le moteur proactif pourrait devenir un goulot d'étranglement. Deuxièmement, les performances générales du moteur sont cruciales pour obtenir un système le plus réactif possible, aussi bien pour stocker des données dans le PIP, que pour agir sur l'environnement grâce aux actionneurs.

Pour y remédier, bien que le matériel utilisé comme environnement d'exécution du moteur est un facteur tout aussi important, nous pourrions également rajouter un ou plusieurs moteurs proactifs, s'occupant d'une seule partie des appareils IoT, nous assurant ainsi de ne pas surcharger les moteurs proactifs.

Conclusion

Nous avons commencé ce mémoire en établissant l'état des connaissances existantes. Il a d'abord été question du contrôle d'accès, d'un point de vue général et ensuite à travers les différents modèles existants. Cela a permis de mieux cerner leur importance en matière de sécurité et leur usage dans les systèmes de contrôle d'accès, ainsi que de définir les principaux atouts fournis par le modèle ABAC, notamment sa grande flexibilité et ses politiques offrant une plus grande granularité. Le standard XACML a donné une base commune et concrète au modèle ABAC, en définissant une architecture et un langage permettant de définir des politiques.

Ensuite, les concepts de proactive computing et d'autonomic computing ont été introduits comme deux visions de ce que pourrait être le futur de l'informatique. Celles-ci prennent place dans un monde où le nombre d'appareils qui nous entourent est devenu exorbitant, et où l'informatique pourrait évoluer vers quelque chose de moins interactif, mais plus autonome et proactif, où les ordinateurs pourraient fonctionner en étant supervisés par l'homme. La réalisation du proactive computing a également été expliquée à travers le moteur proactif, établi sur un système à base de règles et développé à l'Université du Luxembourg.

Pour terminer l'état des connaissances existantes, c'est le paradigme de l'Internet des Objets qui a été examiné à travers 3 visions. L'IoT a surtout permis de mettre en avant la notion de « context-awareness », faisant ainsi le lien avec le proactive computing. En effet, cette notion fait référence aux informations caractérisant l'état d'une entité physique, que le proactive computing pourrait utiliser pour fonctionner.

La problématique autour de laquelle ce mémoire était axé est la combinaison du modèle ABAC avec un système IoT, dans le but d'améliorer et d'exploiter au maximum le premier. En effet, nous avons vu que l'application d'un contrôle d'accès combiné à un système IoT pouvait théoriquement faire une combinaison puissante. Et pour la rendre pragmatique, nous avons choisi d'utiliser un moteur proactif, en tant que middleware entre le modèle ABAC et un système IoT. Ainsi, cette problématique a été divisée en deux parties, la première visant à élaborer la façon de faire remonter les données des capteurs vers le moteur proactif, pour les traiter et les rendre accessibles au référencement dans les politiques de sécurité. La seconde partie visait à procéder dans le sens contraire, c'est-à-dire à comment utiliser le moteur proactif pour agir sur l'environnement lorsque cela est nécessaire.

Pour y répondre, une architecture logique a été explorée, plaçant le moteur proactif entre le PIP de l'architecture définie par XACML et le système IoT composé de capteurs et d'actionneurs. Après une exploration des différents protocoles

disponibles dans le domaine de l'IoT, nous avons choisi d'améliorer le moteur en y ajoutant le support du protocole MQTT, un protocole « machine à machine » permettant de récupérer les données des capteurs et d'envoyer des commandes aux actionneurs. Grâce au concept de scénario, un ensemble structuré de règles du moteur proactif, nous avons montré comment il était possible de traiter ces données et de les stocker dans une base de données commune au moteur proactif et au PIP. Ainsi, ces données traitées pouvaient être référencées en tant qu'attributs dans les politiques de sécurité évaluées par le PDP. De plus, les scénarios exécutés par le moteur proactif disposaient de la possibilité d'envoyer des messages MQTT aux actionneurs pour les commander et donc agir sur l'environnement.

Nous avons ensuite montré par un exemple concret, dans le cadre d'un centre de données modulaire virtuel, mais proche de la réalité, comment il était possible définir un scénario récupérant les températures provenant de différents capteurs, pour ensuite traiter ces données afin de détecter des valeurs anormales pouvant être synonymes d'un problème. Une valeur agrégée de la température était ensuite stockée, toujours grâce à une règle du scénario, dans la base de données liée au PIP. Un exemple de politique de sécurité référençant cette valeur a également été expliqué.

Ce mémoire a donc permis de montrer que le moteur proactif constituait un moyen efficace pour combiner le modèle ABAC avec un système IoT. La richesse des attributs disponibles se trouve donc grandie, en mettant à disposition des données pouvant être de plus haut niveau que les données brutes, dû à un traitement préalable. De plus, de manière tout aussi importante, l'utilisation du moteur proactif permet aussi d'agir sur l'environnement en envoyant des commandes aux actionneurs.

La suite logique serait de tester notre solution dans un environnement réel, ce qui permettrait de préciser et d'éclaircir les points d'amélioration restants, ainsi que les ajustements encore nécessaires. Il s'agirait donc de répondre aux questions de déploiement de la solution dans un environnement de production, mais également de la configuration des paramètres du moteur proactif pour optimiser son exécution.

Enfin, notons que le travail préalable à la réalisation de ce mémoire avait mené à la rédaction d'une publication scientifique [35], disponible à l'annexe A, reprenant les principales conclusions établies dans cette ultime partie.

Bibliographie

[1] R. S. Sandhu et P. Samarati. « Access control : principle and practice ». Dans : *IEEE Communications Magazine* 32.9 (sept. 1994), p. 40-48. issn : 0163-6804. doi : 10.1109/35.312842.

[2] C. D. P. K. Ramli, H. R. Nielson et F. Nielson. *Modelling and Analysing Access Control Policies in XACML 3.0*. Kgs. Lyngby : Technical University of Denmark (DTU), 2015.

[3] Jean-Noël Colin. « Sécurité et fiabilité des systèmes informatiques ». INFOM119. Université de Namur, 2016. url : https://directory.unamur.be/teaching/courses/IHDCM035/2016.

[4] Chung Tong Hu, David F. Ferraiolo et David R. Kuhn. « Assessment of Access Control Systems ». Dans : *NIST Interagency/Internal Report (NISTIR) - 7316* (sept. 2006). url : https://www.nist.gov/publications/assessment-access-control-systems.

[5] Pierangela Samarati et Sabrina Capitani de Vimercati. « Access Control : Policies, Models, and Mechanisms ». Dans : *Foundations of Security Analysis and Design*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, sept. 2000, p. 137-196. isbn : 978-3-540-42896-1. doi : 10.1007/3-540-45608-2_3. url : https://link.springer.com/chapter/10.1007/3-540-45608-2_3.

[6] David Ferraiolo et D Kuhn. « Role-Based Access Control ». Dans : *ACM Trans. Inf. Syst. Secur.* 4 (sept. 1997).

[7] D. Ferraiolo, R. Kuhn et R. Sandhu. « RBAC Standard Rationale : Comments on "A Critique of the ANSI Standard on Role-Based Access Control" ». Dans : *IEEE Security Privacy* 5.6 (nov. 2007), p. 51-53. issn : 1540-7993. doi : 10.1109/MSP.2007.173.

[8] V. C. Hu, D. R. Kuhn et D. F. Ferraiolo. « Attribute-Based Access Control ». Dans : *Computer* 48.2 (fév. 2015), p. 85-88. issn : 0018-9162. doi : 10.1109/MC.2015.33.

[9] Chung Tong Hu et al. « Guide to Attribute Based Access Control (ABAC) Definition and Considerations ». Dans : *Special Publication (NIST SP) - 800-162* (jan. 2014). url : https://www.nist.gov/publications/guide-attribute-based-access-control-abac-definition-and-considerations.

[10] OASIS Standard. *eXtensible Access Control Markup Language (XACML) Version 3.0*. Jan. 2013. url : http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html.

[11] J. C. R. Licklider. « Man-Computer Symbiosis ». Dans : *IRE Transactions on Human Factors in Electronics* HFE-1.1 (mar. 1960), p. 4-11. issn : 0099-4561. doi : 10.1109/THFE2.1960.4503259.

[12] David Tennenhouse. « Proactive Computing ». Dans : *Communications of the ACM* 43.5 (mai 2000), p. 43-50. issn : 0001-0782. doi : 10.1145/332833.332837.

[13] R. Want, T. Pering et D. Tennenhouse. « Comparing autonomic and proactive computing ». Dans : *IBM Systems Journal* 42.1 (2003), p. 129-135. issn : 0018-8670. doi : 10.1147/sj.421.0129.

[14] J. O. Kephart et D. M. Chess. « The vision of autonomic computing ». Dans : *Computer* 36.1 (jan. 2003), p. 41-50. issn : 0018-9162. doi : 10.1109/MC.2003.1160055.

[15] Denis Zampunieris. « Implementation of a Proactive Learning Management System ». Dans : *Proceedings of E-Learn - World Conference on E-Learning in Corporate, Government, Healthcare & Higher Education*. 2006, p. 3145-3151. isbn : 978-1-880094-60-0. url : http://hdl.handle.net/10993/13857.

[16] Denis Shirnin, Sandro Reis et Denis Zampunieris. « Design of Proactive Scenarios and Rules for Enhanced e-Learning ». Dans : *Proceedings of the 4th International Conference on Computer Supported Education, Porto, Portugal 16-18 April, 2012*. SciTePress – Science et Technology Publications, 2012, p. 253-258. isbn : 978-989-8565-06-8. doi : 10.5220/0003956302530258. url : http://hdl.handle.net/10993/2663.

[17] Gilles Neyens, Remus-Alexandru Dobrican et Denis Zampunieris. « Enhancing Mobile Devices with Cooperative Proactive Computing ». Dans : *Proceedings of the 5th International Conference on Advanced Collaborative Networks, Systems and Applications (COLLA 2015)*. IARIA, oct. 2015. isbn : 978-1-61208-436-7. url : http://hdl.handle.net/10993/22199.

[18] Remus-Alexandru Dobrican, Gilles Irénée Fernand Neyens et Denis Zampunieris. « SilentMeet - A Prototype Mobile Application for Real-time Automated Group-based Collaboration ». Dans : *Proceedings of the 5th International Conference on Advanced Collaborative Networks, Systems and Applications (COLLA 2015)*. IARIA, oct. 2015, p. 52-56. isbn : 978-1-61208-436-7. url : http://hdl.handle.net/10993/22198.

[19] Amy Nordrum. *Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated*. IEEE Spectrum : Technology, Engineering, and Science News. 18 août 2016. url : https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated (visité le 14/04/2018).

[20] Luigi Atzori, Antonio Iera et Giacomo Morabito. « The Internet of Things : A survey ». Dans : *Computer Networks* 54.15 (oct. 2010), p. 2787-2805. issn : 1389-1286. doi : 10.1016/j.comnet.2010.05.010.

[21] Laurent Schumacher. « Informatique ambiante et mobile ». INFOM450. Université de Namur, fév. 2017. url : https://directory.unamur.be/teaching/courses/INFOM450/2016.

[22] Jayavardhana GUBBI et al. « Internet of Things (IoT) : A vision, architectural elements, and future directions ». Dans : *Future Generation Computer Systems*. Including Special sections : Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services & Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond 29.7 (sept. 2013), p. 1645-1660. ISSN : 0167-739X. DOI : 10.1016/j.future.2013.01.010.

[23] Gregory D. ABOWD et al. « Towards a Better Understanding of Context and Context-Awareness ». Dans : *Handheld and Ubiquitous Computing*. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, sept. 1999, p. 304-307. ISBN : 978-3-540-66550-2. DOI : 10.1007/3-540-48157-5_29. URL : https://link.springer.com/chapter/10.1007/3-540-48157-5_29.

[24] C. PERERA et al. « Context Aware Computing for The Internet of Things : A Survey ». Dans : *IEEE Communications Surveys Tutorials* 16.1 (2014), p. 414-454. ISSN : 1553-877X. DOI : 10.1109/SURV.2013.042313.00197.

[25] Jasmin GUTH et al. « A Detailed Analysis of IoT Platform Architectures : Concepts, Similarities, and Differences ». Dans : *Internet of Everything : Algorithms, Methodologies, Technologies and Perspectives*. Springer, 2018, p. 81-101. DOI : 10.1007/978-981-10-5861-5_4.

[26] Vitor ROZSA et al. « An Application Domain-Based Taxonomy for IoT Sensors ». Dans : *23rd ISPE International Conference on Transdisciplinary Engineering : Crossing Boundaries*. Curitiba, Brazil, oct. 2016. DOI : 10.3233/978-1-61499-703-0-249. URL : https://hal.archives-ouvertes.fr/hal-01581127.

[27] *IEEE 802.15.4*. URL : http://www.ieee802.org/15/pub/TG4.html (visité le 05/05/2018).

[28] S. AUST, R. V. PRASAD et I. G. M. M. NIEMEGEERS. « IEEE 802.11ah : Advantages in standards and further challenges for sub 1 GHz Wi-Fi ». Dans : *2012 IEEE International Conference on Communications (ICC)*. 2012 IEEE International Conference on Communications (ICC). Juin 2012, p. 6885-6889. DOI : 10.1109/ICC.2012.6364903.

[29] M. SIEKKINEN et al. « How low energy is bluetooth low energy ? Comparative measurements with ZigBee/802.15.4 ». Dans : *2012 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. 2012 IEEE Wireless Communications and Networking Conference Workshops (WCNCW). Avr. 2012, p. 232-237. DOI : 10.1109/WCNCW.2012.6215496.

[30] Rogelio Reyna GARCIA. « Understanding the Zigbee stack ». Dans : *ZigBee alliance home page* (2006).

[31] Mathilde DURVY et al. « Making Sensor Networks IPv6 Ready ». Dans : *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems*. SenSys '08. New York, NY, USA : ACM, 2008, p. 421-422. ISBN : 978-1-59593-990-6. DOI : 10.1145/1460412.1460483. URL : http://doi.acm.org/10.1145/1460412.1460483 (visité le 05/05/2018).

[32] Ammar Rayes et Samer Salam. « IoT Protocol Stack : A Layered View ». Dans : *Internet of Things From Hype to Reality*. Springer, Cham, 2017, p. 93-138. isbn : 978-3-319-44858-9 978-3-319-44860-2. doi : 10.1007/978-3-319-44860-2_5. url : https://link.springer.com/chapter/10.1007/978-3-319-44860-2_5 (visité le 06/05/2018).

[33] Vasileios Karagiannis et al. « A survey on application layer protocols for the internet of things ». Dans : *Transaction on IoT and Cloud Computing* 3.1 (2015), p. 11-17.

[34] OASIS Standard. *MQTT Version 3.1.1*. Déc. 2015. url : http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.

[35] Noé Picard, Jean-Noël Colin et Denis Zampunieris. « Context-aware and Attribute-based Access Control Applying Proactive Computing to IoT System ». Dans : *Proceedings of the workshop on Security, Privacy, Big Data and Internet of Things SPBDIoT 2018 : Within the 3rd International Conference on Internet of Things, Big Data and Security - IoTBDS 2018*. Mar. 2018. url : https://researchportal.unamur.be/fr/publications/context-aware-and-attribute-based-access-control-applying-proacti.

Annexe A

SPBDIoT 2018

À la fin du stage de recherche à l'Université du Luxembourg, nous avons réalisé un article scientifique visant à présenter les différents résultats obtenus. Ainsi, cet article reprend plus brièvement les différents aspects de la solution telle que décrite à travers ce mémoire, élaborée dans le but de combiner le modèle ABAC et l'Internet des Objets.

L'article a été soumis pour révision à une session spéciale, la *SPBDIoT – Special Session on Recent Advances on Security, Privacy, Big Data and Internet of Things*, de la 3ème conférence internationale sur l'Internet des Objets, des Big data et de la sécurité. Il a par la suite été accepté pour une présentation orale de 30 minutes, qui s'est déroulée le 19 mars 2018 sur l'île de Madère au Portugal. La version finale de cet article, tel qu'il a été publié, est disponible ci-après.

Context-aware and Attribute-based Access Control applying Proactive Computing to IoT system

Noé Picard[1], Jean-Noël Colin[1] and Denis Zampunieris[2]
[1]*PReCISE Research Center, University of Namur, Belgium*
[2]*FSTC, University of Luxembourg, Luxembourg*
noe.picard@student.unamur.be, jean-noel.colin@unamur.be, denis.zampunieris@uni.lu

Keywords: Internet of Things, Access Control, ABAC, Event Analysis, Proactive Computing

Abstract: ABAC allows for high flexibility in access control over a system through the definition of policies based on attribute values. In the context of an IoT-based system, these data can be supplied through its sensors connected to the real world, allowing for context-awareness. However, the ABAC model alone does not include proposals for implementing security policies based on verified and/or meaningful values rather than on raw data flowing from the sensors. Nor does it allow to implement immediate action on the system when some security flaw is detected, while this possibility technically exists if the system is equipped with actuators next to its sensors. We show how to circumvent these limitations by adding a proactive engine to the ABAC components, that runs rule-based scenarios devoted to sensor data pre-processing, to higher-level information storage in the PIP, and to real-time, automatic reaction on the system through its actuators when required.

1 INTRODUCTION

Access control is an important part of today systems. Whether it is about physical access control or software access control, it has become an essential and critical element for most businesses. Over the years, multiple models have emerged, like the relatively recent model called Attribute-based access control (ABAC). The concept is not especially innovative, but with the rise of Internet of Things (IoT) systems, the ABAC model could turn out to be very interesting. In fact, IoT sensors allow for monitoring and collecting data about the environment on which access control might apply. Moreover, IoT actuators provide a way for the system to interact with the physical world. With all the data that such a system can provide, the ABAC policies could only be enhanced (Rath and Colin, 2017).

However, some concerns arise when one tries to apply those two concepts together. To make the data coming from the sensors available to the ABAC model architecture, a gap between two components has to be filled, as we will demonstrate in Section 3, and no straightforward solution could be found. Ideally, the envisioned system would also interact with the surrounding environment. Still, how to make this interaction easier without implementing a completely new system? These are the issues that we want to address with the help of proactive computing. We introduce a way to use the proactive computing paradigm, through a proactive engine, to enhance the ABAC model with an IoT system.

The rest of this paper is structured as follows. In Section 2, the access control part is explained along the ABAC model. Section 3 describes the consequences of applying access control to an IoT system. Section 4 explains how the proactive computing makes this application easier. The resulting architecture, which derives from the ABAC model, is described in Section 5. To illustrate how the proactive engine would work in a such case, the Section 6 provides a direct use case example. Section 7 describes how it can be extended to real working systems. Finally, Section 8 completes this paper with our conclusions.

2 ATTRIBUTE-BASED ACCESS CONTROL

Access control is used to define what a subject can do or which resources it can access. Several models for access control have been proposed in the literature like, e.g., discretionary access control, mandatory access control or role-based access control. In this pa-

per, we are concerned with the Attribute-Based Access Control (ABAC) model. For an introduction to ABAC, see for example (Hu et al., 2014).

The main goal of ABAC is to provide high flexibility in access control. For instance, it considers that access control does not always only concern the identity and related roles of the subject trying to access a resource. Indeed, additional information could influence the access decision like the current state of the subject, its actions or the environment. These data are referenced as attributes in the ABAC terminology and attributes are the building blocks of access control policies. Within those policies, attributes can be combined in complex expressions.

2.1 ABAC Model Architecture

The underlying architecture to the ABAC model is generally composed of three main components. The Policy Enforcement Point (PEP) protects the services, data, etc. on which access control is required. When a subject wants to access a resource, the PEP intercepts this request and translates it in an authorization request understandable by the Policy Decision Point (PDP).

The PDP is at the core of the architecture as is the component that takes the final decision (Deny/Permit) regarding a request. This node often relies on two sub-components. One is the Policy Repository Point (PRP) which is responsible for making policies available to the PDP. The other one is the Policy Administration Point (PAP) that acts as the interface for system administrators that allows them to create, edit or delete policies.

Finally, the Policy Information Point (PIP) allows the PDP to retrieve the current attribute values that are needed for the computation of the policies. The rest of this paper is mainly concerned by discussions and proposals around the PIP.

2.2 XACML Standard

XACML (OASIS Standard, 2013) stands for eXtensible Access Control Markup Language. It has been created as a standard by defining a language and a protocol to convey information about access control. It is mainly based on ABAC, but it can also be specialized for other access control models like RBAC. The XACML language gives the possibility to define policies using XML notations. The standard proposes a computation model to evaluate policies against access requests which is based on the ABAC architecture described in Section 2.1.

3 APPLYING ACCESS CONTROL TO IOT

One of the strengths of ABAC is the possibility to use a set of multiple and diverse attributes in the policies. It is even more interesting if this set can be supplied with data from an IoT system, because it can provide lots of information through its sensors connected to the real world.

3.1 Main Benefits From This Combination

As a simple example, one could think of a room equipped with connected sensors that monitor temperature, humidity, presence, and so on in the room. If we store the data received by those devices in a database available through the PIP, then we could write XACML policies that allow or not the access to the room (supposing that there are also connected actuators that (un)lock the room doors, see Section 3.2 and 4) that rely on the current contextual information in the room.

A second additional benefit is the possibility to control how the database behind the PIP is supplied with flows of data. In XACML, there is no recommended way to implement this database, so it allows us to imagine a solution that processes the data coming from the sensors before to store information in the PIP. This way, by avoiding to use raw values from the sensors that may not be relevant and/or by aggregating several values into upper-level data, we could enhance the coherence and the pertinence of the attributes available to the set of policies.

3.2 Limits in Its Implementation

While ABAC offers great extensibility and flexibility compared to other access control models, developers will face some of its limits when it comes to implement it in a real-world system.

For instance, knowing how to fill the attribute values backing up the PIP might be as simple as inserting information in a database, but who is responsible to collect and store the data? It is not really a limit of the ABAC model that we underline here, as it takes into account the diversity of the attributes sources behind the PIP by using it as an interface between the sources and the PDP. But the concern is that if we heavily base the access policies on environmental attributes, it might be cumbersome to implement systems that fill the sources of the PIP.

Also, assuming one is using ABAC with an IoT system, it surely desires to interact with the environ-

ment through the sensors and the actuators. But there is no way to be able to interact synchronously with the environment just by using ABAC. One might say that there is the mechanism of obligations, defined in XACML as operations that should be performed by the PEP when they are returned alongside the authorization decision. But this only allows the PEP to have an effect on the environment when a subject tries to access a resource, not when something has changed within the environment.

4 PROACTIVE COMPUTING AT HELP

Before talking about how to address the limits laid down in Section 3.2, using proactive computing, let us first introduce it. In an article, Tennenhouse established theoretically the basics of proactive computing (Tennenhouse, 2000). He described it in regards with his vision of the future of computing, namely the transition from "human-centered to human-supervised (or even unsupervised) computing". In other words, human-centered computing can be referenced as interactive computing, where a system being used by a user is locked to wait for user actions. However, with proactive computing, the human is no longer placed in the loop, but above it.

4.1 Proactive Engine

At the University of Luxembourg, Zampunieris D. and his team had developed a proactive engine that follows the premises of proactive computing as described by Tennenhouse: "working on behalf of, or *pro*, the user, and acting on their own initiative". Without entering too much in the detail of the implementation, there are some key concepts to understand.

The engine is a *Rule-running system (RRS)*, it executes rules at a certain frequency. With the concepts of *Rule*, one can create a *Scenario* which is a dynamic set of rules obeying some path.

Furthermore, a *Rule* is the basic element in the engine and it can be subdivided in five stages. (1) The first one is called *Data acquisition*. It is during this stage that all the data, necessary to the proper execution of the following stages, is gathered. (2) The *activation guards* stage acts like a trigger for the third and fourth stages. It can be any type of boolean expression. (3) The purpose of the third stage, *Conditions*, is to offer the possibility to perform more in-depth tests on the context. (4) To effectively do something, there is the *Actions* phase, which provides a lot of freedom as well as power, given that the system is implemented in the Java programming language. (5) Finally, the *Rule Generation* step concludes the execution of a rule and allows the creation of other rules, or to clone itself to stay in the system.

To store the rules to be executed at each iteration (or at a further one if it is impossible to execute all of them in one iteration), the engine relies on a FIFO queue. For more information about the other aspects of the engine, see (Zampunieris, 2006).

4.2 Make the Combination Easier

In the Section 3.2, we laid down some limits of the ABAC model, especially when it is applied to an IoT system. These can be easily taken away using a proactive engine in an effective way. In fact, a proactive engine could fill the gap between an ABAC architecture and an IoT system.

On the one hand, there is the concern about the way to make attribute values, coming from sensors data, available to the PDP. Of course, it is the purpose of the PIP to make those available, but as stated before, it is just an interface between some data sources and the PDP. The idea is to use some rules (ideally multiple scenarios) to supply a particular source holding sensors data. But it would not just be raw data, as it is possible to implement scenarios that somehow (see Section 6 for an example) analyze, compute, combine, etc. this data. Therefore, we can assure a certain coherence and completeness in what we store as attributes.

Before going further, note that the scenarios must not be mistaken with Artificial Intelligence (AI), those are predefined, nevertheless they can be multiple and complex. Their sequencing is decided over a period of time based on the events that have occurred.

Then, on the other hand, proactive computing helps a lot when it is necessary to react on the actuators of the IoT system. Like before, specific scenarios could stay tuned for new events that require action(s) and react as quickly as possible.

5 ENHANCED ABAC MODEL ARCHITECTURE

To better understand how a proactive engine can handle his two main tasks (assigned in Section 4.2), this section explains the logical architecture that we came up with. In the Figure 1, there are 3 important parts: the access control part, the proactive engine and the IoT system.

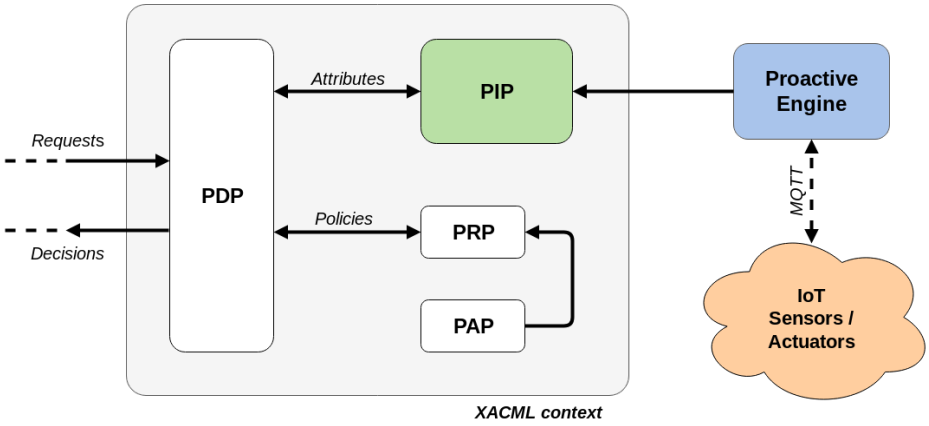

Figure 1: Enhanced ABAC model architecture

5.1 Comply with the Standard

The access control part is delimited by the *XACML context* frame. It follows the architecture described in 2.1, the PDP receive the authorization requests and deliver the decisions. To do so, it retrieves the policies applicable to the current request and evaluates them. If the policies contains references to attributes not available in the direct context – which will be almost always the case because we want to rely exclusively on environmental attributes, the PDP asks the PIP for the corresponding values.

Note that the PEP component is intentionally not represented on the Figure 1, because usually there will be several of them and it can be in many forms and at different places across a system.

5.2 Enhanced Proactive Engine

The proactive engine described in Section 4.1 lacks of one important feature: the communication with the IoT system. Therefore, we chose to implement alongside the engine a MQTT broker. MQTT (Message Queuing Telemetry Transport) is a protocol based on the *Publish-subscribe* pattern, where senders do not know who they are sending messages to, but instead, publish messages on topics. Similarly, the receivers express their interest for a certain topic without knowing anything about the senders. For detailed information about the MQTT protocol, see (OASIS Standard Incorporating Approved Errata 01, 2015). In this way, a rule inside the engine can subscribe to a certain topic and thereby listen for new messages. This specific type of rule has a buffer where the messages can pile up to be processed afterwards.

Moreover, the proactive engine is connected to a database, implicitly represented by the PIP on the Fig. 1. This allows any rule to insert or retrieve data from it. The schema of the database is left to the developers, as it depends on the type of the information contained in the MQTT messages, and therefore the type of the sensors used.

5.3 Collect Data and Act on the Environment

The final important part is the IoT system (composed of sensors and actuators). As hinted in Section 5.2, the sensors can send their data (using specific topics) to the MQTT broker embedded in the proactive engine. The other way around, for actuators to receive commands, the rules can also send messages through the broker.

6 USE CASE EXAMPLE

To illustrate how our architecture works, this section includes a straightforward example. Consider a simple server room equipped with temperature sensors and a system to cool the place that we can utilize as an actuator. The scenario – from the proactive engine point of view – represented at the Figure 2 will monitor the temperature and take actions.

The sensors send their data to the topics `sensors/temperature/X`, where `X` is the identification number of the sensor. A rule in the proactive engine, named `SENSORS_TEMPERATURE_MONITORING`

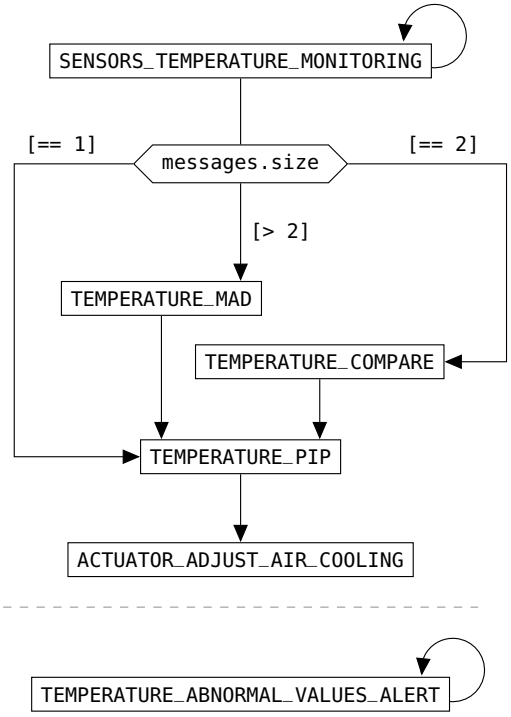

Figure 2: Use case example - Scenario structure

for example, could subscribe to the topic filter `sensors/temperature/+`, meaning that this rule will receive data from all temperature sensors. This rule will always analyse the latest twenty messages received, by clearing its messages buffer in the *actions* step if necessary. If the messages buffer is empty, the rule is not activated. If it has been, it creates other rules according to the number of messages. Note that this rule has a specific type or class inside the engine, we could call it a *MQTT rule*. This type of rule is automatically cloned in the queue in order to always listen for new MQTT messages. The following pseudo code describes its logic:

```
data acquisition: /

activation guards:
  messages.size() > 0

conditions: /

actions:
  if (messages.size() > N_TEMPERATURES_TAKEN
      )
    messages.subList(0, (messages.size() -
        1) - N_TEMPERATURES_TAKEN).clear()

  tempsToAnalyze = messages

rules generation:
  if (getActivated())
    size = tempsToAnalyze.size()
    if (size = 1)
      createRule(new TEMPERATURE_PIP(
          tempsToAnalyze.get(0)))
    else if (size = 2)
      createRule(new TEMPERATURE_COMPARE(
          tempsToAnalyze))
    else
      createRule(new TEMPERATURE_MAD(
          tempstoanalyze))
```

From the `SENSORS_TEMPERATURE_MONITORING` rule, there is one of the three following rules created. If only one message was received, the created one is the `TEMPERATURE_PIP` rule. Its purpose is to store in the PIP the `temperature` value passed as a parameter. To do so, it verifies if it is necessary to store the value, i.e. if the difference between the current stored value in the PIP and the value to store is greater than a particular constant (`MINIMUM_DELTA`). If not, the rule is not activated and nothing is stored. Moreover, if the difference is greater than another constant (`MAXIMUM_DELTA`) and that the last temperature update in the PIP was close enough, the value is considered as abnormal and a counter is incremented. This rule is an example of how the proactive engine can solve the first limit presented in Section 3.2. In the final step, the rule `ACTUATOR_ADJUST_AIR_COOLING` is created to take actions on the environment through the cooling system by triggering some actuators.

```
data acquisition:
  previousTemp = getPip().getTemperature()
  lastTempUpdate = getPip().
      getLastTemperatureUpdate()

activation guards: /

conditions:
  Math.abs(currentTemp - previousTemp) >
      MINIMUM_DELTA

actions:
  if ((lastTempUpdate + INTERVAL) >
      currentTime() && Math.abs(currentTemp
       - previousTemp) > MAXIMUM_DELTA)
    getEngineDB().
        incrementAbnormalTemperature
       ValuesCounter()

  getPip().updateTemperature(temperature)

rules generation:
  createRule(new ACTUATOR_ADJUST_AIR_COOLING
      (temperature))
```

The rule `ACTUATOR_ADJUST_AIR_COOLING` ad-

justs the air-cooling system according to the `temperature` value passed as parameter. Here, one can observe that this rule solves the second problem developed in Section 3.2 by acting on the environment.

```
data acquisition: /

activation guards: /

conditions: /

actions:
  if (temperature < LIMIT_MIN) {
    setAirCoolingLevel(AirCoolingLevel.LOW)
  } else if (temperature > LIMIT_MAX) {
    setAirCoolingLevel(AirCoolingLevel.HIGH)
  } else {
    if ((temperature - LIMIT_MIN) <= 2) {
      setAirCoolingLevel(AirCoolingLevel.LOW
          )
    } else if (Math.abs(temperature -
        LIMIT_MAX) <= 2) {
      setAirCoolingLevel(AirCoolingLevel.
          HIGH)
    } else {
      // Between LIMIT_MIN and LIMIT_MAX
      setAirCoolingLevel(AirCoolingLevel.
          MODERATE)
    }
  }

rules generation: /
```

The rule `TEMPERATURE_COMPARE` is created by the rule `SENSORS_TEMPERATURE_MONITORING` when the messages buffer has a size of 2. It compares two temperatures values passed as arguments. If the difference between the two is not too high, the mean is store. But it can also increment the abnormal value counter if this difference is in fact too high.

When there are strictly more than 2 messages in the buffer, it is the rule `TEMPERATURE_MAD` that is created. This one uses a statistics measure called Median Absolute Deviation (MAD) to detect any abnormal value. If one is actually detected, as before, a counter is incremented.

These two last rules create the `TEMPERATURE_PIP` when it is necessary to store a temperature value in the PIP. Finally, the rule `TEMPERATURE_ABNORMAL_VALUES_ALERT` is always present in the engine queue, because it watches over the abnormal value counter and generates an alert if this counter exceeds some threshold.

7 INNOVATIVE, PROACTIVE, ABAC-BASED SECURITY

The example in Section 6 might seem simplistic, but it was essential to introduce and illustrate the main idea of this paper. However, it does not represent all the extent of the model that we developed. In fact, extending it to a real working system is achievable and has been already performed in a controlled testbed.

At the University of Luxembourg, a simulator representing the IoT system of a "modular data center" had been conceived to test a proactive engine with multiple scenarios. It was mainly simulating the sensors and the actuators of the system, to provide a kind of sandbox in order to experiment with the proactive engine. The results were conclusive, and prove that the engine could work on larger system, due to its ability to run several scenarios in parallel, in order to handle multiple sensors and actuators.

More globally, the advantages of our enhanced ABAC-based security model become clearer: First, as seen in Section 6, the proactive engine can take the role of filling automatically the PIP with sensors data. The access policies can be more abstract, that is to say the data stored as an attribute could have been pre-processed, thus it is not necessary to be burdened with low-level data in those policies. Secondly, the actuators can be triggered spontaneously by the scenarios, all it requires is implementing rules that interact with any of the actuators. Moreover, the coherence of attributes can also be insured if the scenarios are well-defined. Finally, another key aspect is the separation of concerns that allows the division in scenarios. With this concept it is possible to focus on one type of data or actuators at a time, which also allows a better maintainability and extendability.

8 CONCLUSIONS

Attribute-based access control applied to IoT system can theoretically make a powerful combination. To make it concrete, two limits were encountered: how to effectively link the sensors data with attribute values in the PIP and how to automatically and immediately act on the environment when it is needed. In order to overcome these, the proactive computing paradigm reflected in a proactive engine turned out to be an ideal choice.

With the concept of proactive scenario, we showed how to provide data coming from the sensors as a source of attributes for the ABAC model. Moreover, in these scenarios, the data can be pre-processed and not just be raw values that a sensor provides,

hence allowing higher level data as attributes. Equally important is the ability to take actions on the environment by sending commands from the proactive engine to the IoT actuators.

This way of doing can be applicable to large systems with numerous sensors and actuators, as the engine can run multiple and complex scenarios in parallel, without performance issues. We believe that this new application of proactive computing opens some promising road towards taking full advantage of the ABAC model and the Internet of Things at the same time.

REFERENCES

Hu, C. T., Ferraiolo, D. F., Kuhn, D. R., Schnitzer, A., Sandlin, K., Miller, R., and Scarfone, K. (2014). Guide to Attribute Based Access Control (ABAC) Definition and Considerations. *Special Publication (NIST SP) - 800-162*.

OASIS Standard (2013). eXtensible Access Control Markup Language (XACML) Version 3.0. `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html`.

OASIS Standard Incorporating Approved Errata 01 (2015). MQTT Version 3.1.1 Plus Errata 01. `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html`.

Rath, T. M. A. and Colin, J. N. (2017). Adaptive Risk-aware Access Control Model for Internet of Things. In *Proceedings of the International Workshop on Secure Internet of Things, in conjunction with Esorics2017*, Oslo.

Tennenhouse, D. (2000). Proactive Computing. *Communications of the ACM*, 43(5):43–50.

Zampunieris, D. (2006). Implementation of a Proactive Learning Management System. In *Proceedings of "E-Learn - World Conference on E-Learning in Corporate, Government, Healthcare & Higher Education"*, pages 3145–3151.